



**UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE MÉXICO**



FACULTAD DE CIENCIAS

**“APLICACIÓN DE REDES NEURONALES DE
APRENDIZAJE PROFUNDO PARA LA SOLUCIÓN DE
LA ECUACIÓN NO LINEAL DE SCHRÖDINGER CON
POTENCIAL LINEAL”**

TESIS

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

P R E S E N T A:

FÍS. FÉLIX ENRIQUE MÉNDEZ ZÚÑIGA

DIRECTORES DE TESIS:

TUTOR ACADÉMICO: Dr. MÁXIMO AUGUSTO AGÜERO GRANADOS

TUTOR EXTERNO ADJUNTO: Dr. LEONID SERKIN

TUTORA ADJUNTA: Dra. TATYANA BELYAEVA LEONIDOVNA

Índice general

Resumen	4
1 Introducción	11
1.1. Contexto y motivación	11
1.2. Objetivos	13
1.3. Justificación	14
2 Marco Teórico	15
2.1. Redes Neuronales Artificiales (ANN)	15
2.2. Redes Neuronales Informadas por la Física (PINN)	38
2.3. Ecuación de Schrödinger No Lineal (NLSE)	39
2.4. Aprendizaje profundo basado en la física: Solucion basada en datos de la ecuacion diferencial parcial no lineal de Shrodin- ger (ref. [2])	43
3 Metodología	63
3.1. Formulación matemática de la NLSE con potenciales externos.	63
3.2. Implementación del Método PINN	73
4 Resultados y Discusión	78
4.1. Soluciones	78
4.2. Análisis General	95
4.3. Comparación de Resultados	97
4.4. Discusión de Resultados	98
4.5. Conclusiones Parciales	103
5 Conclusiones	104
A Programa PINN	105

Resumen

El proyecto de investigación se centra en la aplicación de redes neuronales de aprendizaje profundo (específicamente, las redes neuronales informadas por la física, PINNs) para resolver la ecuación no lineal de Schrödinger (NLSE) con potencial lineal. A lo largo de lo estudiado en la investigación de maestría, se han realizado avances significativos en el estudio teórico, la implementación de métodos numéricos y la optimización de las PINNs para resolver esta ecuación.

- Resultados:

- Se revisó la bibliografía sobre métodos teóricos para resolver la NLSE.
- Se obtuvieron soluciones exactas tipo solitón (1-solitón y 2-solitón) y se graficaron.
- Se confirmó que la NLSE con potencial lineal es completamente integrable y tiene soluciones tipo solitón.
- Se estudiaron y programaron PINNs para resolver varias ecuaciones diferenciales.
- Se analizaron parámetros de aprendizaje como capas neuronales, iteraciones de optimización y cantidad de datos de entrenamiento.
- Se obtuvieron soluciones numéricas para la NLSE y la ecuación de Hirota, con errores aceptables en tiempos de entrenamiento razonables.
- Se utilizaron CPUs comunes para entrenar las redes, demostrando que las PINNs pueden ser eficaces incluso sin hardware especializado.

-Se implementó una PINN para resolver la NLSE con potencial lineal, considerando una solución compleja con dispersión exponencial y un potencial lineal.

-Se encontró que la complejidad de la solución dificultó el aprendizaje de la red, obteniendo errores mayores a los esperados.

-Se comparó el rendimiento entre CPUs y GPUs, demostrando que las GPUs son significativamente más rápidas para el entrenamiento.

-Se obtuvieron errores de aprendizaje dentro de un rango aceptable, pero no se alcanzaron los niveles óptimos observados en otros estudios.

•Conclusiones:

El método AKNS y las transformaciones de Bäcklund son herramientas poderosas para resolver ecuaciones no lineales y obtener soluciones exactas, lo que sirvió como base para comparar con soluciones numéricas obtenidas mediante redes neuronales.

Las PINNs son una herramienta prometedora para resolver ecuaciones diferenciales no lineales, aunque se requiere optimizar los parámetros de aprendizaje para mejorar la precisión y eficiencia.

Aunque se avanzó en la implementación de las PINNs, la complejidad de la solución requirió ajustes adicionales en los parámetros de aprendizaje. Se optimizó el modelo, enfocándose en la segmentación temporal de los datos y en la mejora de los parámetros de la PINN.

Estructura de la tesis

En el capítulo 1 se establece el contexto científico y se justifica la relevancia de la investigación.

Por lo que se explican desafíos en la solución numérica de la NLSE con potenciales externos, destacando limitaciones de métodos tradicionales (FEM, FDM) en problemas no lineales complejos.

Siendo necesarios enfoques innovadores como las PINNs para manejar condiciones de frontera dinámicas y no linealidades fuertes.

Transición al Capítulo 2: Se introduce el marco teórico necesario para abordar el problema, vinculando conceptos de ANN y física matemática.

En el Capítulo 2: Marco Teórico el propósito es brindar fundamentos teóricos de ANN, PINNs y la NLSE.

Tal que las secciones abarcan:

- Sección 2.1: Redes Neuronales Artificiales:

- *Evolución histórica: Desde el perceptrón (Rosenblatt, 1958) hasta arquitecturas profundas modernas.

- *Arquitecturas clave:

- Redes Feedforward: Base para PINNs, con énfasis en propagación hacia adelante y cálculo de gradientes.

- RNN y CNN: Mención breve de su uso en problemas espacio-temporales (ej.: series de tiempo en NLSE).

*Entrenamiento:

-Backpropagation y optimizadores (Adam, L-BFGS-B).

-Función de pérdida MSE y regularización física (Physics-Informed Loss).

●Sección 2.2: PINNs

*Diferenciación automática: Ventaja sobre métodos basados en mallas para ecuaciones diferenciales.

*Aplicaciones documentadas: Ejemplos exitosos en fluidodinámica y mecánica cuántica (referencia a [2]).

*Limitaciones: Sensibilidad a hiperparámetros y coste computacional en problemas de alta dimensión.

●Sección 2.3: NLSE y Solitones

Formulación matemática: Deducción desde principios físicos (ej.: efecto Kerr en óptica).

*Solitones no autónomos:

-Propiedades únicas bajo potenciales externos (adaptación de amplitud y fase).

-Soluciones exactas mediante método AKNS y transformada de Bäcklund (Figuras 3.1–3.2).

Transición al Capítulo 3: Se describe cómo los fundamentos teóricos se traducen en una metodología concreta para resolver la NLSE.

En el Capítulo 3: Metodología el proposito es detallar el diseño e implementación de la PINN para la NLSE.

Tal que las secciones abarcan:

●Sección 3.1: Formulación Matemática

NLSE con potencial lineal:

-Ecuación (3.11): Discusión de términos (dispersión $D(t)$, no linealidad $R(t)$, potencial $\lambda(t)$)

Método AKNS:

-Construcción de pares de Lax y condiciones de integrabilidad (Ecuaciones 3.5–3.9).

Transformada de Bäcklund:

-Generación de soluciones 1 y 2-soliton (Ecuaciones 3.23–3.27).

●Sección 3.2: Implementación de la PINN

Arquitectura:

-Configuración de capas (4 capas ocultas, 100 neuronas por capa) y función de activación ($\tanh$).

-Preprocesamiento de datos: Muestreo LHS para puntos iniciales/frontera ($N_0=50$, $N_b = 50$)

Pérdidas:

- $MSE_{física}$ (Ecuación 2.47): Incorporación de la NLSE en la función de pérdida.

Balance entre MSE_0 (condiciones iniciales) y MSE_b (frontera).

Optimización:

-Protocolo de entrenamiento (Adam + L-BFGS-B).

-Criterios de parada (iteraciones máximas, tolerancia de error).

Transición al Capítulo 4: Los resultados validan la metodología propuesta, comparando predicciones con soluciones analíticas.

En el Capítulo 4: Resultados y Discusión el propósito es presentar y analizar los resultados numéricos, destacando hallazgos clave.

Tal que las secciones abarcan:

●Sección 4.1: Soluciones Obtenidas

Caso base (solitón simple):

- Error L_2 relativo de $1,04 \times 10^{-1}$ (Figura 4.2).
- Análisis de componentes real/imaginaria (Figura 4.3).

Efecto de parámetros:

- Velocidad: Aumento del error al introducir $\kappa_{01} \neq 0$ (Figura 4.5 vs. 4.8).
- Potencial lineal: Impacto en la trayectoria predicha (Figura 4.10).

Comparación de configuraciones:

- Reducción del error al simplificar términos no lineales (Tabla comparativa implícita).

●Sección 4.2: Desempeño Computacional

Tiempos de entrenamiento:

- Rango de 3.5 a 6.9 horas (CPU i5-12400), dependiendo de la complejidad.

Convergencia:

- Gráficas de pérdida (Figuras 4.4, 4.7) mostrando estancamiento en $\sim 10^{-3}$

Discusión crítica:

- Factores que limitan la precisión (ej. : "chirp" autoinducido).
- Ventajas de PINNs sobre métodos espectrales en flexibilidad, pero desafíos en coste computacional.

Transición al Capítulo 5: Síntesis de contribuciones y lecciones aprendidas.

En el Capítulo 5: Conclusiones el propósito es resumir logros y implicaciones.

Hallazgos principales:

- Las PINNs son viables para resolver NLSE con potencial lineal, logrando errores $\sim 10^{-2}$ en casos óptimos.
- La velocidad y el potencial externo son los parámetros que más degradan el rendimiento (errores hasta $2,66 \times 10^{-1}$).

Contribuciones:

- Implementación de un pipeline reproducible para NLSE con códigos en

Python/TensorFlow.

-Caracterización sistemática de hiperparámetros en problemas no autónomos.

Capítulo 1

Introducción

1.1. Contexto y motivación

La ecuación no lineal de Schrödinger (NLSE) es una de las ecuaciones más importantes en física matemática y tiene aplicaciones en una amplia variedad de campos, como la óptica no lineal, los condensados de Bose-Einstein (BEC), la propagación de ondas en plasmas y la hidrodinámica. Esta ecuación describe la evolución de ondas no lineales y dispersivas, y es conocida por admitir soluciones especiales llamadas solitones, que son ondas localizadas que mantienen su forma durante la propagación.

En particular, la NLSE con potencial lineal es un modelo que surge en contextos donde se considera la influencia de un potencial externo, como en sistemas ópticos con fibras no lineales o en la dinámica de partículas en trampas magnéticas. Resolver esta ecuación de manera exacta es un desafío matemático, y aunque existen métodos analíticos como el método de dispersión inversa (AKNS) y las transformaciones de Bäcklund, estos métodos pueden volverse complejos cuando se trata de soluciones no autónomas o con condiciones iniciales y de frontera complicadas.

Por otro lado, el aprendizaje profundo y las redes neuronales artificiales (ANN) han revolucionado la forma en que se abordan problemas complejos en ciencia e ingeniería. En particular, las redes neuronales informadas por la física (PINNs) han surgido como una herramienta poderosa para resolver ecuaciones diferenciales parciales no lineales (PDEs) de manera numérica.

Estas redes combinan el poder de las ANN con el conocimiento físico de las ecuaciones, lo que permite encontrar soluciones aproximadas sin necesidad de mallas computacionales tradicionales.

La motivación principal de este proyecto es explorar y desarrollar métodos innovadores para resolver la NLSE con potencial lineal utilizando técnicas de aprendizaje profundo, específicamente las PINNs. Aunque existen métodos analíticos y numéricos tradicionales para resolver esta ecuación, estos pueden ser limitados en términos de flexibilidad y escalabilidad, especialmente cuando se trata de soluciones complejas o no autónomas.

Las PINNs ofrecen una alternativa prometedora porque:

1. No dependen de mallas: A diferencia de los métodos numéricos tradicionales, las PINNs no requieren mallas estructuradas; trabajan con puntos de colocación, lo que las hace más flexibles para problemas con geometrías complejas o condiciones de frontera no estándar.

2. Incorporan conocimiento físico: Al integrar las ecuaciones físicas directamente en la red neuronal, las PINNs pueden aprender soluciones que respeten las leyes físicas subyacentes, lo que mejora la precisión y generalización del modelo.

3. Escalabilidad: Las PINNs pueden ser entrenadas en hardware moderno (como GPUs), lo que permite resolver problemas de mayor escala en tiempos razonables.

4. Fácil análisis de que tan bueno o malo es el error obtenido: En el contexto PINN se suelen tener referencias de que tan bueno o malo es el error relativo esto depende de los problemas teniendo que:

-Si el error es aproximadamente 10^{-1} significa que hay 10 % de error esto puede considerarse "aceptable" en problemas muy complejos o multidimensionales, pero no es muy bueno en problemas simples.

-Si el error es aproximadamente 10^{-2} significa un error del 1 % en la mayoría de artículos de PINNs, esto ya es reportado como "muy bueno".

-Si el error relativo L^2 es aproximadamente 10^{-3} o menor a esto , esto es que hay .1 % o menos de error, esto es considerado como exelente y mas si es obtenido de un problema complejo.

Además, este proyecto tiene un impacto potencial en varias áreas:

Óptica no lineal: La NLSE modela la propagación de pulsos de luz en fibras ópticas, y su solución es crucial para el diseño de sistemas de comunicación óptica.

Física de materia condensada: En los condensados de Bose-Einstein, la NLSE describe la dinámica de ondas de materia, y su solución es esencial para entender fenómenos como la formación de solitones y la dinámica de partículas en trampas magnéticas.

Plasmas y fluidos: La NLSE también aparece en la descripción de ondas en plasmas y en la propagación de ondas de gravedad en océanos, lo que tiene aplicaciones en geofísica y astrofísica.

1.2. Objetivos

El objetivo principal de este proyecto es investigar y desarrollar métodos basados en PINNs para resolver la NLSE con potencial lineal, comparando las soluciones obtenidas con las soluciones exactas conocidas. Esto permitirá no solo validar la eficacia de las PINNs en este contexto, sino también explorar su potencial para resolver otras ecuaciones diferenciales no lineales en física y matemáticas aplicadas.

El objetivo se desarrolla por los siguientes objetivos secundarios:

1. Estudiar y obtener soluciones exactas de la ecuación no lineal de Schrödinger (NLSE) con potencial lineal utilizando métodos teóricos como el método AKNS y las transformaciones de Bäcklund.
2. Estudiar y aplicar redes neuronales informadas por la física (PINNs) para resolver ecuaciones diferenciales parciales no lineales, incluyendo la NLSE.

3. Aplicar las PINNs para resolver la NLSE con potencial lineal, específicamente para una solución tipo solitón no autónomo con una fase autoinducida ("chirp").

1.3. Justificación

Este trabajo es relevante porque:

- Avances en métodos numéricos: Las PINNs representan un enfoque novedoso para resolver PDEs, y su aplicación a la NLSE con potencial lineal contribuye al desarrollo de métodos numéricos más eficientes y flexibles.
- Aplicaciones prácticas: La solución de la NLSE tiene implicaciones directas en áreas como las telecomunicaciones ópticas, la física de partículas y la dinámica de fluidos, lo que puede llevar a avances tecnológicos y científicos.
- Interdisciplinariedad: Este proyecto combina técnicas de física matemática, aprendizaje automático y computación de alto rendimiento, lo que fomenta la colaboración entre disciplinas y abre nuevas líneas de investigación.

En resumen, este proyecto busca innovar en la resolución de ecuaciones diferenciales no lineales utilizando técnicas de aprendizaje profundo, con el fin de abordar problemas complejos en física y matemáticas aplicadas, y contribuir al desarrollo de herramientas computacionales más eficientes y versátiles.

Capítulo 2

Marco Teórico

2.1. Redes Neuronales Artificiales (ANN)

2.1.1 Introducción a las ANN y su evolución.

Las Redes Neuronales Artificiales (ANN) son un modelo computacional inspirado en el funcionamiento del cerebro humano, específicamente en las redes de neuronas biológicas. Estas redes están compuestas por unidades llamadas neuronas artificiales, que se organizan en capas y se conectan entre sí para procesar información y aprender a partir de datos.

- Conceptos básicos:

- Neurona artificial: Es la unidad básica de una ANN. Recibe entradas, las procesa mediante una función de activación y produce una salida.

- Capas: Las ANN suelen tener una capa de entrada, una o varias capas ocultas y una capa de salida.

- Pesos y sesgos: Son parámetros ajustables que la red aprende durante el entrenamiento para minimizar el error en las predicciones.

- Función de activación: Introduce no linealidades en el modelo, permitiendo que la red aprenda patrones complejos (ejemplos: ReLU, sigmoide, tanh).

- Tipos de ANN:

- Perceptrón: La forma más simple de una ANN, con una sola capa.

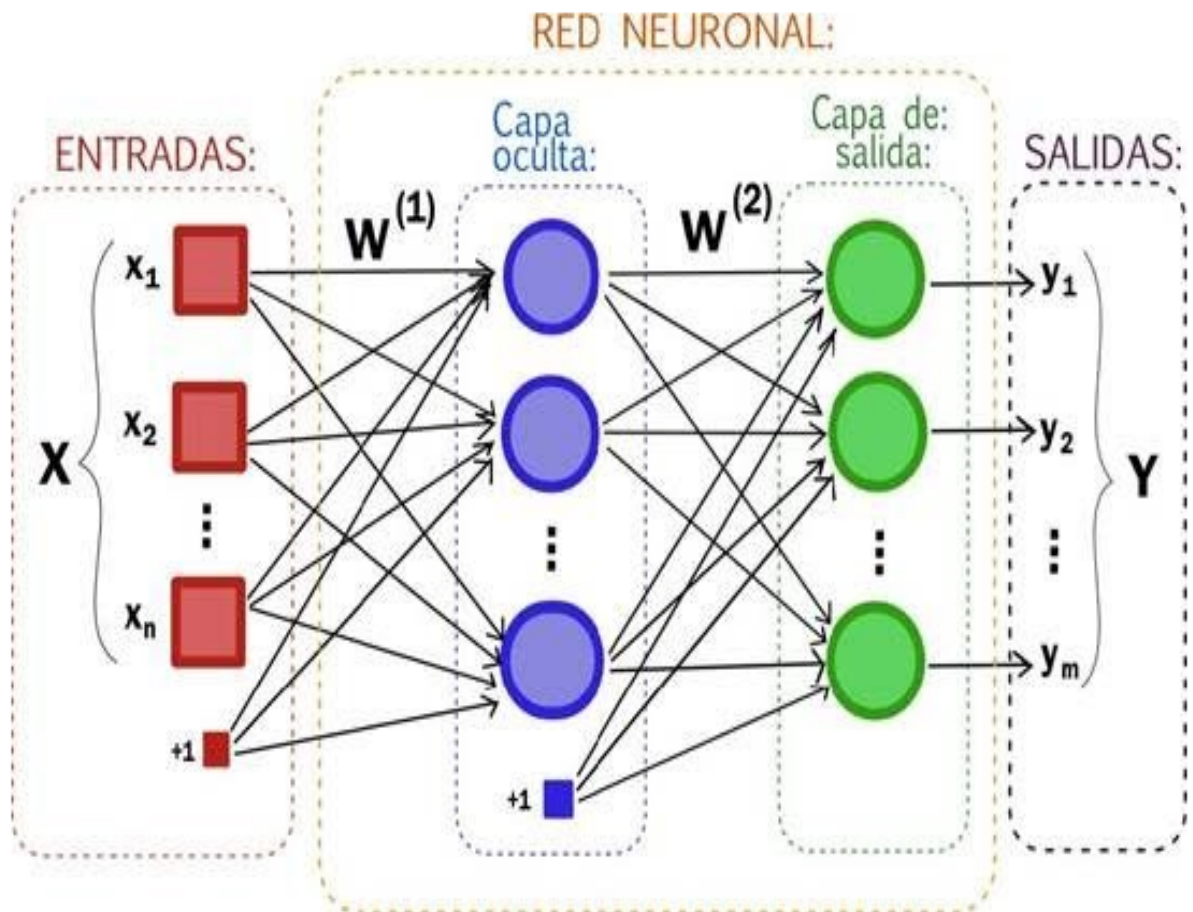


Figura 2.1: Esquema de una red neuronal Fuente: [¿Deep Learning... y clasificación de imágenes?](#).

-Redes Neuronales Feedforward: La información fluye en una sola dirección, desde la entrada hasta la salida.

-Redes Neuronales Recurrentes (RNN): Tienen conexiones recurrentes que permiten procesar secuencias de datos (como series de tiempo o texto).

-Redes Neuronales Convolucionales (CNN): Especializadas en procesar datos tensoriales con estructura espacial/temporal, como imágenes.

- Entrenamiento:

-Aprendizaje supervisado: La red se entrena con datos etiquetados, ajustando los pesos para minimizar el error entre las predicciones y las etiquetas reales.

-Backpropagation: Algoritmo clave para entrenar ANN, que calcula el gradiente del error con respecto a los pesos y los ajusta iterativamente.

Evolución de las ANN.

- Orígenes:

Las primeras ideas sobre ANN surgieron en los años 40 y 50, con el trabajo de Warren McCulloch y Walter Pitts, quienes propusieron un modelo matemático de una neurona.

En 1958, Frank Rosenblatt desarrolló el Perceptrón, uno de los primeros modelos de ANN.

- Décadas de 1980 y 1990:

Se popularizó el uso del algoritmo de backpropagation, lo que permitió entrenar redes con múltiples capas (redes profundas).

A pesar de los avances, las ANN enfrentaron limitaciones debido a la falta de poder computacional y falta de datos suficientes.

- Resurgimiento en el siglo XXI:

Con el aumento del poder computacional (GPUs) y la disponibilidad de grandes cantidades de datos, las ANN experimentaron un renacimiento.

El desarrollo de arquitecturas profundas, como las CNN y las RNN, permitió avances significativos en áreas como visión por computadora y procesamiento de lenguaje natural.

•Tendencias actuales:

Redes Neuronales Profundas (Deep Learning): Las ANN con muchas capas (redes profundas) dominan el campo del aprendizaje automático.

Transformers: Arquitecturas que han revolucionado el procesamiento de lenguaje natural (ejemplo: GPT, BERT).

Aprendizaje no supervisado y por refuerzo: Nuevas técnicas que permiten a las ANN aprender sin datos etiquetados o mediante interacción con un entorno.

Funciones de red de avance (ref. [1])

Los modelos lineales de regresión y clasificación se basan en combinaciones lineales de funciones base no lineales fijas $\phi_j(x)$ y toman la forma

$$y(x, w) = f\left(\sum_{j=1}^M w_j \phi_j(x)\right) \quad (2.1)$$

donde $f(\cdot)$ es una función de activación no lineal. El objetivo es extender este modelo haciendo que las funciones base $\phi_j(x)$ dependan de parámetros y luego permitir que estos parámetros se ajusten, junto con los coeficientes $\{w_j\}$, durante el entrenamiento. Existen muchas formas de construir funciones base no lineales paramétricas. Las redes neuronales utilizan funciones base que siguen la misma forma que (2.1), de modo que cada función base es en sí misma una función no lineal de una combinación lineal de las entradas, donde los coeficientes en la combinación lineal son parámetros adaptativos. Esto lleva al modelo básico de red neuronal, que puede describirse como una serie de transformaciones funcionales. Se construye primero M combinaciones lineales de las variables de entrada $x_1, \dots, x_D$ en la forma

$$a_j = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \quad (2.2)$$

donde $j = 1, \dots, M$ y el superíndice (1) indica que los parámetros correspondientes están en la primera ‘capa’ de la red. Los parámetros $w_{ji}^{(1)}$ son pesos y los parámetros $w_{j0}^{(1)}$ son sesgos. Las cantidades a_j se conocen como

activaciones. Cada uno de ellos se transforma luego utilizando una función de activación no lineal diferenciable $h(\cdot)$ para dar

$$z_j = h(a_j) \quad (2.3)$$

Estas cantidades corresponden a las salidas de las funciones base en (2.1) que, en el contexto de las redes neuronales, se denominan unidades ocultas. Las funciones no lineales $h(\cdot)$ se eligen generalmente como funciones sigmoideas, como la sigmoidea logística o la función ‘tanh’. Siguiendo (2.1), estos valores se combinan nuevamente linealmente para dar activaciones de la unidad de salida.

$$a_k = \sum_{j=1}^M w_{kj}^{(2)} z_j + w_{k0}^{(2)} \quad (2.4)$$

donde $k = 1, \dots, K$ y K es el número total de salidas. Esta transformación corresponde a la segunda capa de la red, y nuevamente los $w_{k0}^{(2)}$ son parámetros de sesgo. Finalmente, las activaciones de la unidad de salida se transforman utilizando una función de activación apropiada para obtener un conjunto de salidas de red y_k . La elección de la función de activación está determinada por la naturaleza de los datos y la distribución supuesta de las variables objetivo y sigue las mismas consideraciones que para los modelos lineales. Por lo tanto, para los problemas de regresión estándar, la función de activación es la identidad tal que $y_k = a_k$. De manera similar, para problemas de clasificación binaria múltiple, cada activación de unidad de salida se transforma utilizando una función sigmoidea logística de modo que

$$y_k = \sigma(a_k) \quad (2.5)$$

donde

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \quad (2.6)$$

Finalmente, para problemas multiclase, una función de activación softmax de la forma

$$p(C_k|x) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \quad (2.7)$$

que se conoce como exponencial normalizada y puede considerarse como una generalización multiclase del sigmoide logístico.

Se pueden combinar estas diversas etapas para obtener la función de red global que, para funciones de activación de unidad de salida sigmoidea, toma la forma

$$y_k(x, w) = \sigma \left(\sum_{j=1}^M w_{kj}^{(2)} h \left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \quad (2.8)$$

donde el conjunto de todos los parámetros de peso y sesgo se han agrupado en un vector w . Por lo tanto, el modelo de red neuronal es simplemente una función no lineal de un conjunto de variables de entrada $\{x_i\}$ a un conjunto de variables de salida $\{y_k\}$ controlado por un vector w de parámetros ajustables.

Esta función se puede representar en forma de diagrama de red como se muestra en la Figura 2.2.

El proceso de evaluación (2.8) puede entonces interpretarse como una propagación hacia adelante de información a través de la red.

Los parámetros de sesgo en (2.2) se pueden absorber en el conjunto de parámetros de peso definiendo una variable de entrada adicional x_0 cuyo valor está fijado en $x_0 = 1$, por lo que (2.2) toma la forma

$$a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i \quad (2.9)$$

De manera similar, se pueden absorber los sesgos de la segunda capa en los pesos de la segunda capa, de modo que la función de red general se convierta en

$$y_k(x, w) = \sigma \left(\sum_{j=0}^M w_{kj}^{(2)} h \left(\sum_{i=0}^D w_{ji}^{(1)} x_i \right) \right) \quad (2.10)$$

Como se puede ver en la Figura 2.2, el modelo de red neuronal comprende dos etapas de procesamiento, cada una de las cuales se asemeja al modelo de perceptrón y por esta razón la red neuronal también se conoce como perceptrón multicapa, o MLP. Sin embargo, una diferencia clave en comparación con el perceptrón es que la red neuronal utiliza no linealidades sigmoideas continuas en las unidades ocultas, mientras que el perceptrón

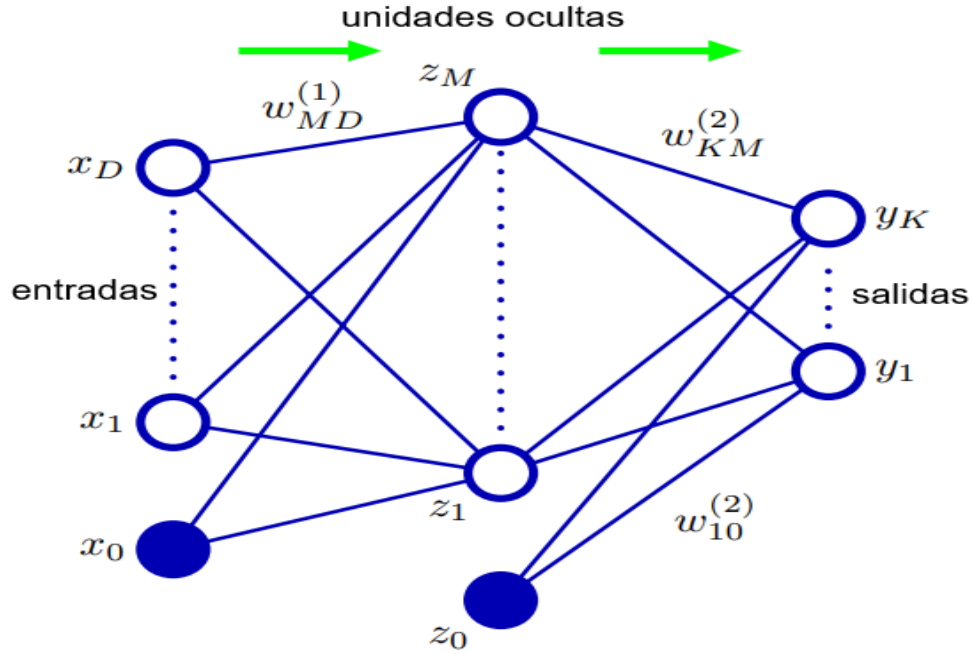


Figura 2.2: Diagrama de red para la red neuronal de dos capas correspondiente a (2.8). Las variables de entrada, ocultas y de salida están representadas por nodos, y los parámetros de peso están representados por enlaces entre los nodos, en los que los parámetros de sesgo están denotados por enlaces que provienen de variables de entrada y variables ocultas x_0 y z_0 . Las flechas indican la dirección del flujo de información a través de la red durante la propagación hacia adelante. Fuente: Pattern Recognition and Machine Learning, Christopher M. Bishop.

utiliza no linealidades de función escalonada. Esto significa que la función de la red neuronal es diferenciable con respecto a los parámetros de la red, y esta propiedad jugará un papel central en el entrenamiento de la red. Si las funciones de activación de todas las unidades ocultas en una red se toman como lineales, entonces para cualquier red de este tipo siempre podemos encontrar una red equivalente sin unidades ocultas. Esto se desprende del hecho de que la composición de transformaciones lineales sucesivas es en sí misma una transformación lineal. Sin embargo, si el número de unidades ocultas es menor que el número de unidades de entrada o de salida, entonces

las transformaciones que la red puede generar no son las transformaciones lineales más generales posibles de entradas a salidas porque se pierde información en la reducción de dimensionalidad en las unidades ocultas.

La arquitectura de red que se muestra en la Figura 2.2 es la más utilizada en la práctica. Sin embargo, se puede generalizar fácilmente, por ejemplo, considerando capas adicionales de procesamiento, cada una de las cuales consiste en una combinación lineal ponderada de la forma (2.4) seguida de una transformación elemento por elemento utilizando una función de activación no lineal.

Otra generalización de la arquitectura de red es incluir conexiones de capa de salto, cada una de las cuales está asociada con un parámetro adaptativo correspondiente. Por ejemplo, en una red de dos capas, estos irían directamente de las entradas a las salidas. En principio, una red con unidades ocultas sigmoideas siempre puede imitar conexiones de capa de salto (para valores de entrada acotados) utilizando un peso de primera capa suficientemente pequeño como para que, en su rango operativo, la unidad oculta sea efectivamente lineal, y luego compensando con un valor de peso grande desde la unidad oculta hasta la salida. En la práctica, sin embargo, puede ser ventajoso incluir conexiones de capa de salto explícitamente.

Además, la red puede ser dispersa y no estar presentes todas las conexiones posibles dentro de una capa. Debido a que existe una correspondencia directa entre un diagrama de red y su función matemática, podemos desarrollar asignaciones de red más generales considerando diagramas de red más complejos. Sin embargo, estas deben restringirse a una arquitectura de propagación hacia adelante, es decir, a una que no tenga ciclos dirigidos cerrados, para garantizar que las salidas sean funciones deterministas de las entradas. Cada unidad (oculta o de salida) en dicha red calcula una función dada por

$$z_k = h\left(\sum_j w_{kj} z_j\right) \quad (2.11)$$

donde la suma se ejecuta sobre todas las unidades que envían conexiones a la unidad k (y se incluye un parámetro de sesgo en la suma). Para un conjunto dado de valores aplicados a las entradas de la red, la aplicación sucesiva de (2.11) permite evaluar las activaciones de todas las unidades de la red, incluidas las de las unidades de salida.

Por lo tanto, se dice que las redes neuronales son aproximadores universales.

Por ejemplo, una red de dos capas con salidas lineales puede aproximar uniformemente cualquier función continua en un dominio de entrada compacto con precisión arbitraria, siempre que la red tenga una cantidad suficientemente grande de unidades ocultas. Este resultado es válido para una amplia gama de funciones de activación de unidades ocultas, pero excluye los polinomios.

La capacidad de una red de dos capas para modelar una amplia gama de funciones se ilustra en la Figura 2.3. Esta figura también muestra cómo las unidades ocultas individuales trabajan en colaboración para aproximarse a la función final. El papel de las unidades ocultas en un problema de clasificación simple se ilustra en la Figura 2.4 utilizando datos de clasificación sintéticos.

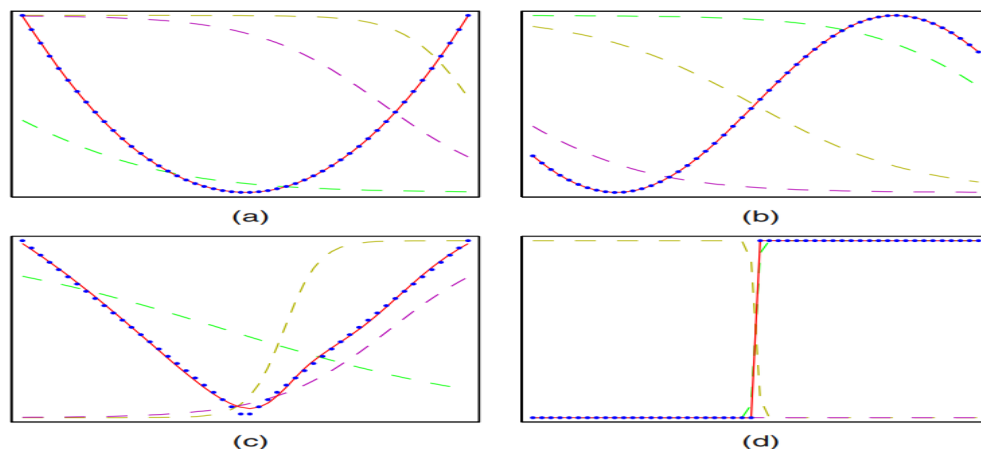


Figura 2.3: Ilustración de la capacidad de un perceptrón multicapa para aproximarse a cuatro funciones diferentes que comprenden (a) $f(x) = x^2$, (b) $f(x) = \text{sen}(x)$, (c) $f(x) = |x|$ y (d) $f(x) = H(x)$ donde $H(x)$ es la función paso Heaviside. En cada caso los puntos de datos son $N = 50$, los puntos azules se muestran como muestreados uniformemente en x durante el intervalo $(-1,1)$ y se evaluaron los valores correspondientes de $f(x)$. Estos puntos de datos se utilizan luego para entrenar una red de dos capas que tiene tres unidades ocultas con funciones de activación "tanh" y unidades de salida lineal. Las funciones de red resultantes se muestran mediante las curvas rojas y las salidas de las tres unidades ocultas se muestran mediante las tres curvas discontinuas. Fuente: Pattern Recognition and Machine Learning, Christopher M. Bishop.

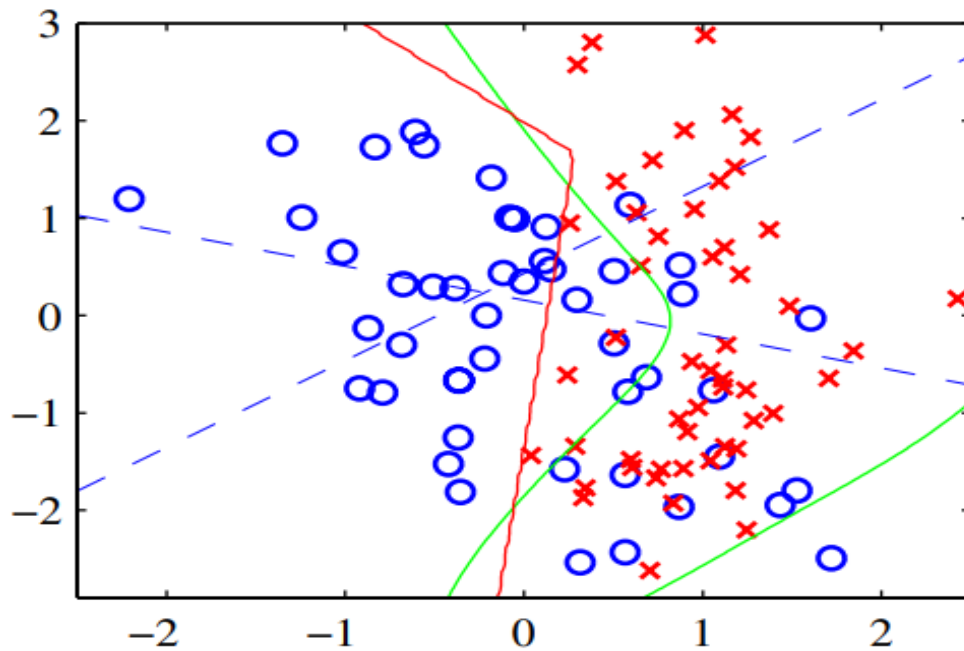


Figura 2.4: Ejemplo de la solución de un problema simple de clasificación de dos clases que involucra datos sintéticos utilizando una red neuronal que tiene dos entradas, dos unidades ocultas con funciones de activación 'tanh' y una única salida que tiene una función de activación sigmoidea logística. Las líneas azules discontinuas muestran los contornos $z = 0,5$ para cada una de las unidades ocultas, y la línea roja muestra la superficie de decisión $y = 0,5$ para la red. A modo de comparación, la línea verde indica el límite de decisión óptimo calculado a partir de las distribuciones utilizadas para generar los datos. Fuente: Pattern Recognition and Machine Learning, Christopher M. Bishop.

2.1.1.1 Entrenamiento de red

Dado un conjunto de entrenamiento que comprende un conjunto de vectores de entrada $\{x_n\}$, donde $n = 1, \dots, N$ junto con un correspondiente conjunto de vectores de objetivo $\{t_n\}$, minimizando la función error

$$E(w) = \frac{1}{2} \sum_{n=1}^N \|y(x_n, w) - t_n\|^2 \quad (2.12)$$

Sin embargo, se puede proporcionar una visión mucho más general del entrenamiento de la red dando primero una interpretación probabilística a las salidas de la red.

Suponemos que t tiene una distribución gaussiana con una media dependiente, que viene dada por la salida de la red neuronal, de modo que

$$p(t|x, w) = \mathcal{N}(t|y(x, w), \beta^{-1}) \quad (2.13)$$

donde β es la precisión (varianza inversa) del ruido gaussiano. Para la distribución condicional dada por (2.13), es suficiente tomar como función de activación de la unidad de salida la identidad, porque dicha red puede aproximar cualquier función continua de x a y . Dado un conjunto de datos de N observaciones independientes, distribuidas de forma idéntica $X = \{x_1, \dots, x_N\}$, junto con los valores objetivo correspondientes $t = \{t_1, \dots, t_N\}$ se puede construir la función de probabilidad correspondiente

$$p(t|X, w, \beta) = \prod_{n=1}^N p(t_n|x_n, w, \beta) \quad (2.14)$$

Tomando el logaritmo negativo, se obtiene la función de error

$$\frac{\beta}{2} \sum_{n=1}^N \{y(x_n, w) - t_n\}^2 - \frac{N}{2} \ln(\beta) + \frac{N}{2} \ln(2\pi) \quad (2.15)$$

la cual puede ser usada para aprender los parámetros w y β . Teniendo en cuenta que en la literatura sobre redes neuronales, es habitual considerar la minimización de una función de error en lugar de la maximización de la verosimilitud (logaritmo), por lo que aquí se usará esta convención. Considerando primero la determinación de w . Maximizar la función de verosimilitud es equivalente a minimizar la función de error de suma de cuadrados dada por

$$E(w) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, w) - t_n\}^2 \quad (2.16)$$

donde se descartaron constantes aditivas y multiplicativas. El valor de w encontrado al minimizar $E(w)$ se denotará w_{ML} porque corresponde a la solución de máxima verosimilitud. En la práctica, la no linealidad de la función de red $y(x_n, w)$ hace que el error $E(w)$ no sea convexo, por lo que en la práctica se pueden encontrar máximos locales de la probabilidad, correspondientes a mínimos locales de la función de error.

Una vez encontrado w_{ML} , el valor de β se puede encontrar minimizando la probabilidad logarítmica negativa para obtener

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^N \{y(x_n, w_{ML}) - t_n\}^2 \quad (2.17)$$

Tomando en cuenta que esto se puede evaluar una vez que se complete la optimización iterativa necesaria para encontrar w_{ML} . Si tenemos múltiples variables objetivo y asumimos que son independientes condicionalmente en x y w con precisión de ruido compartida β , entonces la distribución condicional de los valores objetivo está dada por

$$p(t|x, w) = \mathcal{N}(t|y(x, w), \beta^{-1}I) \quad (2.18)$$

Siguiendo el mismo argumento que para una única variable objetivo, vemos que los pesos de máxima verosimilitud se determinan minimizando la función de error de suma de cuadrados (2.12). La precisión del ruido se da entonces por

$$\frac{1}{\beta_{ML}} = \frac{1}{NK} \sum_{n=1}^N \|y(x_n, w_{ML}) - t_n\|^2 \quad (2.19)$$

donde K es el número de variables objetivo. El supuesto de independencia puede abandonarse a expensas de un problema de optimización ligeramente más complejo.

En el caso de regresión, se ve la red como si tuviera una función de activación de salida que es la identidad, de modo que $y_k = a_k$. La función de error de suma de cuadrados correspondiente tiene la propiedad

$$\frac{\partial E}{\partial a_k} = y_k - t_k \quad (2.20)$$

Considerando el caso de clasificación binaria en el que una única variable objetivo t tal que $t = 1$ denota la clase $\mathcal{C}_1$ y $t = 0$ denota la clase $\mathcal{C}_2$. Considerando una red que tiene una única salida cuya función de activación es una sigmoidea logística.

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)} \quad (2.21)$$

de modo que $0 \leq y(x, w) \leq 1$. Se puede interpretar $y(x, w)$ como la probabilidad condicional $p(\mathcal{C}_1|x)$, con $p(\mathcal{C}_2|x)$ dado por $1 - y(x, w)$. La distribución condicional de objetivos dadas las entradas es entonces una distribución de Bernoulli de la forma

$$p(t|x, w) = y(x, w)^t \{1 - y(x, w)\}^{1-t} \quad (2.22)$$

Considerando un conjunto de entrenamiento de observaciones independientes, entonces la función de error, que viene dada por la probabilidad logarítmica negativa, es entonces una función de error de entropía cruzada de la forma

$$E(w) = - \sum_{n=1}^N \{t_n \ln(y_n) + (1 - t_n) \ln(1 - y_n)\} \quad (2.23)$$

donde y_n denota $y(x_n, w)$. Teniendo en cuenta que no existe un análogo de la precisión del ruido β porque se supone que los valores objetivo están correctamente etiquetados. Sin embargo, el modelo se puede ampliar fácilmente para permitir errores de etiquetado. Simard et al. (2003) descubrieron que el uso de la función de error de entropía cruzada en lugar de la suma de cuadrados para un problema de clasificación conduce a un entrenamiento más rápido y a una generalización mejorada.

Teniendo K clasificaciones binarias separadas para realizar, entonces se puede usar una red que tenga K salidas, cada una de las cuales tiene una función de activación sigmoidea logística. A cada salida se asocia una etiqueta de clase binaria $t_k \in \{0, 1\}$, donde $k = 1, \dots, K$. Asumiendo que las etiquetas de clase son independientes, dado el vector de entrada, entonces la distribución condicional de los objetivos es

$$p(t|x, w) = \prod_{k=1}^K y_k(x, w)^{t_k} [1 - y_k(x, w)]^{1-t_k} \quad (2.24)$$

Tomando el logaritmo negativo de la función de probabilidad correspondiente se obtiene la siguiente función de error

$$E(w) = - \sum_{n=1}^N \sum_{k=1}^K \{t_{nk} \ln(y_{nk}) + (1 - t_{nk}) \ln(1 - y_{nk})\} \quad (2.25)$$

donde y_{nk} denota $y_k(x_n, w)$. Nuevamente, la derivada de la función de error con respecto a la activación para una unidad de salida particular toma la forma (2.20) tal como en el caso de regresión.

Suponiendo que se utiliza una red estándar de dos capas del tipo que se muestra en la Figura 2.2. Se ve que los parámetros de peso en la primera capa de la red se comparten entre las distintas salidas, mientras que en el modelo lineal cada problema de clasificación se resuelve independientemente. La primera capa de la red puede considerarse como la realización de una extracción de características no lineal, y el hecho de compartir características entre las diferentes salidas puede ahorrar cálculos y también puede conducir a una generalización mejorada.

Por último, considerando el problema de clasificación multiclase estándar en el que cada entrada se asigna a una de K clases mutuamente excluyentes. Las variables binarias objetivo $t_k \in \{0, 1\}$ tienen un esquema de codificación 1 de K que indica la clase, y las salidas de la red se interpretan como $y_k(x, w) = p(t_k = 1|x)$, lo que lleva a la siguiente función de error

$$E(w) = - \sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln(y_k(x_n, w)) \quad (2.26)$$

se ve que la función de activación de la unidad de salida, que corresponde al enlace canónico, está dada por la función softmax

$$y_k(x, w) = \frac{\exp(a_k(x, w))}{\sum_j \exp(a_j(x, w))} \quad (2.27)$$

que satisface $0 \leq y_k \leq 1$ y $\sum_k y_k = 1$. Tomando en cuenta que $y_k(x, w)$ no cambia si se agrega una constante a todos los $a_k(x, w)$, lo que hace que la función de error sea constante para algunas direcciones en el espacio de

peso. Esta degeneración se elimina si se agrega un término de regularización apropiado a la función de error.

Una vez más, la derivada de la función de error con respecto a la activación para una unidad de salida particular toma la forma familiar (2.20).

En resumen, existe una elección natural tanto de la función de activación de la unidad de salida como de la función de error de coincidencia, según el tipo de problema que se resuelva. Para la regresión, se utilizan salidas lineales y un error de suma de cuadrados; para las clasificaciones binarias (múltiples independientes), se utilizan salidas sigmoideas logísticas y una función de error de entropía cruzada; y para la clasificación multiclase, se usan salidas softmax con la función de error de entropía cruzada multiclase correspondiente. Para los problemas de clasificación que involucran dos clases, se puede usar una única salida sigmoidea logística o, alternativamente, utilizar una red con dos salidas que tengan una función de activación de salida softmax.

Optimización de parámetros

Encontrar un vector de peso w que minimice la función elegida $E(w)$. En este punto, es útil tener una imagen geométrica de la función de error, que se puede ver como una superficie situada sobre el espacio de peso como se muestra en la Figura 2.5. En primer lugar, teniendo en cuenta que si se da un pequeño paso en el espacio de peso desde w a $w + \delta w$ entonces el cambio en la función error es $\delta E \simeq \delta w^T \nabla E(w)$, donde el vector $\nabla E(w)$ apunta en la dirección de la mayor tasa de aumento de la función de error. Debido a que el error $E(w)$ es una función continua suave de w , su valor más pequeño ocurrirá en un punto en el espacio de pesos tal que el gradiente de la función de error se anule, de modo que

$$\nabla E(w) = 0 \quad (2.28)$$

De lo contrario, se podría dar un pequeño paso en la dirección de $-\nabla E(w)$ y, por lo tanto, reducir aún más el error. Los puntos en los que el gradiente se desvanece se denominan puntos estacionarios y pueden clasificarse además en mínimos, máximos y puntos de silla.

El objetivo es encontrar un vector w tal que $E(w)$ tome su valor más pequeño.

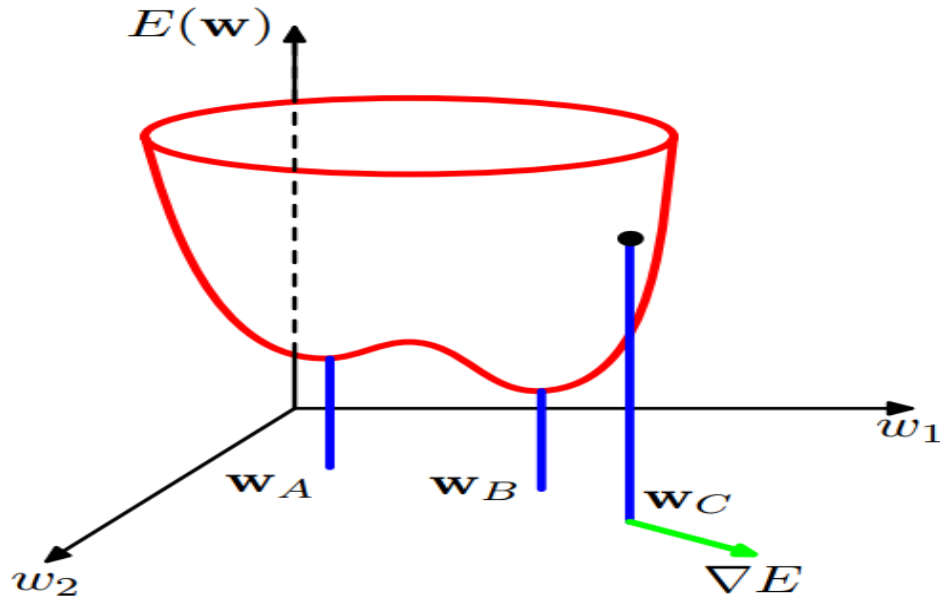


Figura 2.5: Vista geométrica de la función de error $E(w)$ como una superficie situada sobre el espacio de peso. El punto w_A es un mínimo local y w_B es el mínimo global. En cualquier punto w_C , el gradiente local de la superficie de error está dado por el vector ∇E Fuente: Pattern Recognition and Machine Learning, Christopher M. Bishop.

Sin embargo, la función de error normalmente tiene una dependencia altamente no lineal de los pesos y los parámetros de sesgo, por lo que habrá muchos puntos en el espacio de pesos en los que el gradiente desaparece (o es numéricamente muy pequeño). Para cualquier punto w que sea un mínimo local, habrá otros puntos en el espacio de pesos que sean mínimos equivalentes. Por ejemplo, en una red de dos capas del tipo mostrado en la Figura 2.2, con M unidades ocultas, cada punto en el espacio de pesos es miembro de una familia de $M!2^M$ puntos equivalentes.

Además, normalmente habrá múltiples puntos estacionarios desiguales y, en particular, múltiples mínimos desiguales. Un mínimo que corresponde al valor más pequeño de la función de error para cualquier vector de ponderación se denomina mínimo global. Cualquier otro mínimo correspondiente

a valores superiores de la función de error se denomina mínimo local. Para una aplicación exitosa de redes neuronales, puede que no sea necesario encontrar el mínimo global (y, en general, no se sabrá si se ha encontrado), pero sí puede ser necesario comparar varios mínimos locales para encontrar una solución suficientemente buena.

Porque claramente no hay esperanza de encontrar una solución analítica a la ecuación $\nabla E(w) = 0$ se recurre a procedimientos numéricos iterativos. La optimización de funciones no lineales continuas es un problema ampliamente estudiado y existe una extensa literatura sobre cómo resolverlo eficientemente. La mayoría de las técnicas implican elegir un valor inicial $w^{(0)}$ para el vector de pesos y luego moverse a través del espacio de pesos en una sucesión de pasos de la forma

$$w^{(\tau+1)} = w^{(\tau)} + \Delta w^{(\tau)} \quad (2.29)$$

donde τ etiqueta el paso de iteración. Diferentes algoritmos implican diferentes opciones para la actualización del vector de peso $\Delta w^{(\tau)}$. Muchos algoritmos utilizan información de gradiente y, por lo tanto, requieren que, después de cada actualización, el valor de $\nabla E(w)$ se evalúe en el nuevo vector de peso $w^{(\tau+1)}$. Para comprender la importancia de la información del gradiente, es útil considerar una aproximación local a la función de error basada en una expansión de Taylor.

Aproximación cuadrática local

Se puede obtener una idea del problema de optimización y de las distintas técnicas para resolverlo considerando una aproximación cuadrática local a la función de error.

Considerando la expansión de Taylor de $E(w)$ alrededor de algún punto $\hat{w}$ en el espacio de pesos

$$E(W) \simeq E(\hat{w}) + (w - \hat{w})^T b + \frac{1}{2}(w - \hat{w})^T H(w - \hat{w}) \quad (2.30)$$

donde los terminos cubicos y de mayor orden han sido omitidos. Aquí b se define como el gradiente de E evaluado en $\hat{w}$

$$b \equiv \nabla E|_{w=\hat{w}} \quad (2.31)$$

y la matriz Hessiana $H = \nabla \nabla E$ con elementos

$$(H)_{ij} \equiv \left. \frac{\partial E}{\partial w_i \partial w_j} \right|_{w=\hat{w}} \quad (2.32)$$

De (2.30), la aproximación local correspondiente a el gradiente es dada por

$$\nabla E \simeq b + H(w - \hat{w}) \quad (2.33)$$

Para los puntos w que estén suficientemente cerca de $\hat{w}$, estas expresiones darán aproximaciones razonables para el error y su gradiente.

Considerando el caso particular de una aproximación cuadrática local alrededor de un punto w^* esto es un mínimo de la función de error. En este caso no hay término lineal, porque $\nabla E = 0$ en w^* , y (2.30) se hace

$$E(w) = E(w^*) + \frac{1}{2}(w - w^*)^T H(w - w^*) \quad (2.34)$$

donde la Hesiana H es evaluada en w^* . Para interpretar esto geométicamente, se considera la ecuación de valor propio de la matriz hessiana.

$$Hu_i = \lambda_i u_i \quad (2.35)$$

donde los eigenvectores u_i forma un conjunto completo ortonormal de modo que

$$u_i^T u_j = \delta_{ij} \quad (2.36)$$

Expandiendo $(w - w^*)$ como una combinación lineal de eigenvectores de la forma

$$w - w^* = \sum_i \alpha_i u_i \quad (2.37)$$

Esto puede considerarse como una transformación del sistema de coordenadas en el que el origen se traslada al punto w^* y los ejes se rotan para alinearse con los vectores propios (a través de la matriz ortogonal cuyas columnas son u_i). Substituyendo (2.37) en (2.34), y usando (2.35) y (2.36) permite escribir la función de error en la forma

$$E(w) = E(w^*) + \frac{1}{2} \sum_i \lambda_i \alpha_i^2 \quad (2.38)$$

Una matriz H se define positiva si y solo si

$$v^T H v > 0. \quad (2.39)$$

para todo v .

Porque los vectores propios $\{u_i\}$ forman un conjunto completo, un vector arbitrario v puede ser escrito en la forma

$$v = \sum_i c_i u_i \quad (2.40)$$

De (2.35) y (2.36) se tiene

$$v^T H v = \sum_i c_i^2 \lambda_i \quad (2.41)$$

y entonces H será definida positiva si, y sólo si, todos sus valores propios son positivos. En el nuevo sistema de coordenadas, cuyos vectores base están dados por los vectores propios $\{u_i\}$, los contornos de la constante E son elipses centradas en el origen, como se ilustra en la Figura 2.6. Para un espacio de peso unidimensional, un punto estacionario w^* será un mínimo si

$$\left. \frac{\partial^2 E}{\partial w^2} \right|_{w^*} > 0 \quad (2.42)$$

El resultado correspondiente en D -dimensiones es que la matriz hesiana, evaluada en w^* , debería ser definida positiva.

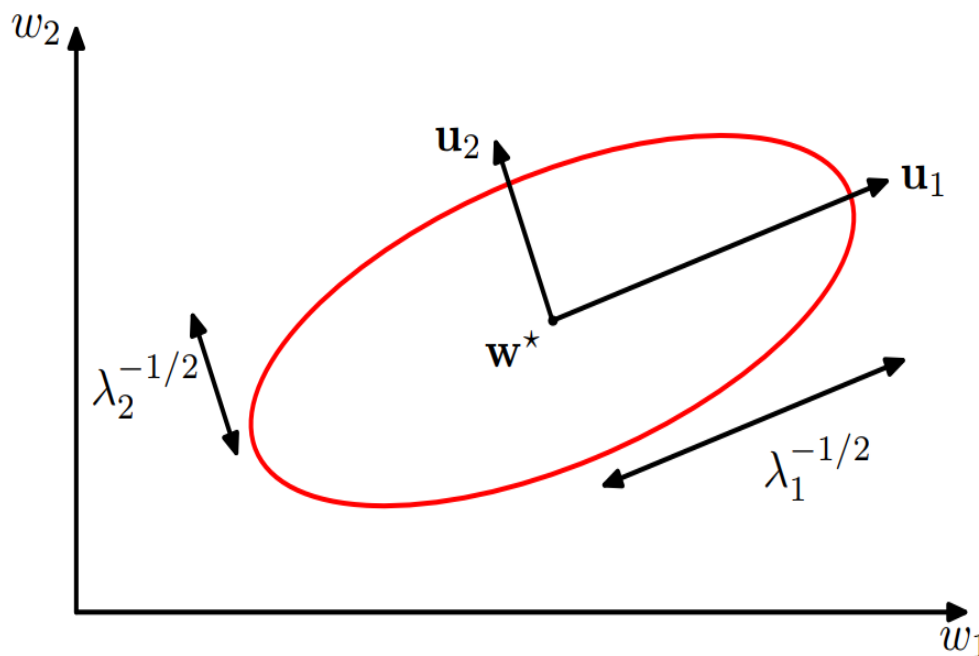


Figura 2.6: En las proximidades de un mínimo w^* , la función de error puede aproximarse mediante una ecuación cuadrática. Los contornos de error constante son entonces elipses cuyos ejes están alineados con los vectores propios u_i de la matriz hessiana, con longitudes inversamente proporcionales a las raíces cuadradas de los valores propios correspondientes λ_i . (ref [1]).

Uso de información de gradiente.

Es posible evaluar eficientemente el gradiente de una función de error mediante el procedimiento de retropropagación. El uso de esta información de gradiente puede mejorar significativamente la velocidad con la que se localizan los mínimos de la función de error. A continuación, se explica por qué. En la aproximación cuadrática a la función de error, dada en (5.28), la superficie de error está especificada por las cantidades b y H , que contienen un total de $W(W + 3)/2$ elementos independientes (porque la matriz H es simétrica), donde W es la dimensionalidad de w (i.e el número total de parámetros adaptativos en la red). Por lo tanto, la ubicación del mínimo de esta aproximación cuadrática depende de $O(W^2)$ parámetros, y no se debería esperar poder localizar el mínimo hasta que se hayan reunido $O(W^2)$ piezas

de información independientes. Si no se utiliza información de gradiente, se esperaría tener que realizar $O(W^2)$ evaluaciones de función, cada una de las cuales requeriría $O(W)$ pasos. Por lo tanto, el esfuerzo computacional necesario para encontrar el mínimo utilizando dicho enfoque sería $O(W^3)$. Comparando esto con un algoritmo que utiliza la información del gradiente. Dado que cada evaluación de ∇E aporta W elementos de información, se podría esperar encontrar el mínimo de la función en $O(W)$ evaluaciones del gradiente. Como se vera, al utilizar la retropropagación de errores, cada evaluación requiere solo $O(W)$ pasos, por lo que el mínimo se puede encontrar en $O(W^2)$ pasos. Por esta razón, el uso de la información de gradiente constituye la base de algoritmos prácticos para el entrenamiento de redes neuronales.

Optimización del descenso de gradiente.

El enfoque más simple para usar la información del gradiente es elegir la actualización de peso en (2.29) para que comprenda un pequeño paso en la dirección del gradiente negativo, de modo que

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \quad (2.43)$$

donde el parametro $\eta > 0$ es conocido como la tasa de aprendizaje. Tras cada actualización, se revalúa el gradiente para el nuevo vector de ponderación y se repite el proceso. Cabe destacar que la función de error se define con respecto a un conjunto de entrenamiento, por lo que cada paso requiere procesar todo el conjunto de entrenamiento para evaluar ∇E . Las técnicas que utilizan todo el conjunto de datos a la vez se denominan métodos por lotes. En cada paso, el vector de ponderación se desplaza en la dirección de la mayor tasa de disminución de la función de error; por lo tanto, este enfoque se conoce como descenso de gradiente o descenso más pronunciado. Aunque este enfoque puede parecer intuitivamente razonable, en realidad resulta ser un algoritmo deficiente.

Para la optimización por lotes, existen métodos más eficientes, como los gradientes conjugados y los métodos cuasi-Newton, que son mucho más robustos y mucho más rápidos que el simple descenso de gradiente. A diferencia del descenso de gradiente, estos algoritmos tienen la propiedad de que la función de error siempre disminuye en cada iteración a menos que el vector de peso haya llegado a un mínimo local o global.

Para encontrar un mínimo suficientemente bueno, puede ser necesario ejecutar un algoritmo basado en gradientes varias veces, cada vez utilizando un punto de inicio diferente elegido aleatoriamente, y comparando el rendimiento resultante en un conjunto de validación independiente.

Sin embargo, existe una versión sobre la línea del descenso de gradiente que ha demostrado ser útil en la práctica para entrenar redes neuronales en grandes conjuntos de datos. Las funciones de error basadas en la máxima verosimilitud para un conjunto de observaciones independientes comprenden una suma de términos, uno para cada punto de datos.

$$E(w) = \sum_{n=1}^N E_n(w) \quad (2.44)$$

El descenso de gradiente en línea, también conocido como descenso de gradiente secuencial o descenso de gradiente estocástico, realiza una actualización del vector de ponderación en función de un punto de datos a la vez, de modo que

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_n(w^{(\tau)}) \quad (2.45)$$

Esta actualización se repite recorriendo los datos, ya sea secuencialmente o seleccionando puntos al azar con reemplazo. Por supuesto, existen escenarios intermedios en los que las actualizaciones se basan en lotes de puntos de datos.

Una ventaja de los métodos sobre línea en comparación con los métodos por lotes es que los primeros manejan la redundancia en los datos de manera mucho más eficiente. Para ilustrarlo, consideremos un ejemplo extremo: tomamos un conjunto de datos y duplicamos su tamaño duplicando cada punto de datos. Tenga en cuenta que esto simplemente duplica la función de error, lo que equivale a usar la función de error original. Los métodos por lotes requieren el doble de esfuerzo computacional para evaluar el gradiente de la función de error por lotes, mientras que los métodos sobre línea no se ven afectados. Otra propiedad del descenso de gradiente sobre línea es la posibilidad de escapar de los mínimos locales, ya que un punto estacionario con respecto a la función de error para todo el conjunto de datos generalmente no será un punto estacionario para cada punto de datos individualmente.

2.1.2 Aplicaciones de las ANN en Problemas Físicos y Matemáticos.

- Solución de Ecuaciones Diferenciales

-Las ANN pueden aproximar soluciones de ecuaciones diferenciales ordinarias (ODE) y ecuaciones diferenciales parciales (PDE), útiles en física e ingeniería; como por ejemplo simulación de flujos de fluidos, propagación de calor, ecuaciones de onda.

-Métodos: Redes neuronales como Physics-Informed Neural Networks (PINNs) incorporan las ecuaciones físicas directamente en la función de pérdida.

- Optimización Matemática

-Las ANN se usan para resolver problemas de optimización no lineal, como encontrar mínimos globales en funciones complejas; como por ejemplo algoritmos como Simulated Annealing combinados con redes neuronales para optimización combinatoria.

- Modelado de Sistemas Físicos

-Predicción de comportamientos en sistemas dinámicos (mecánica clásica, electromagnetismo, física cuántica); como por ejemplo aprendizaje de leyes físicas a partir de datos experimentales usando redes neuronales.

- Análisis de Datos Científicos

-Extracción de patrones en grandes conjuntos de datos físicos (e.g., física de partículas, astronomía); como por ejemplo clasificación de señales en telescopios o aceleradores de partículas.

- Aproximación de Funciones

-Las ANN son aproximadores universales de funciones, capaces de modelar relaciones no lineales en datos matemáticos.

2.2. Redes Neuronales Informadas por la Física (PINN)

Los PINNs (Physics-Informed Neural Networks, o Redes Neuronales Informadas por la Física) son un enfoque innovador que combina el aprendizaje automático con ecuaciones diferenciales basadas en leyes físicas. Son muy útiles para resolver problemas de dinámica de fluidos, transferencia de calor, mecánica de sólidos y más, especialmente cuando los datos experimentales son escasos.

1. ¿Qué son los PINNs?

Los PINNs son redes neuronales entrenadas para resolver ecuaciones diferenciales gobernadas por leyes físicas. A diferencia de los métodos numéricos tradicionales (como FEM o FDM), los PINNs:

- Aprenden soluciones sin requerir una discretización espacial/temporal fina.

- Incorpora las ecuaciones físicas directamente en la función de pérdida.

- Son útiles para problemas inversos (donde se desconocen parámetros del modelo) y directos.

2. Estructura de un PINN

(a) Red Neuronal

Toma como entrada variables independientes (ej. coordenadas x, y, t).

Predice variables dependientes (ej. velocidad u , temperatura T).

(b) Pérdida Física (Physics Loss)

La red debe satisfacer la ecuación diferencial (PDE).

Se calcula el residuo de la PDE y se añade a la función de pérdida.

(c) Pérdida de Datos (Data Loss)

Si hay datos experimentales, se compara la salida de la red con ellos (MSE).

(d) Pérdida de Condiciones Iniciales/Frontera

Se imponen condiciones como $T(x, 0) = f(x)$ o $u(0, t) = g(t)$

(e) Error relativo L^2 relativo es una métrica que mide qué tan cerca está la solución aprendida por la red neuronal de la solución exacta (o de referencia) en términos de la norma L^2 .

Se define:

$$L^2 = \frac{\|u_{pred} - u_{exact}\|_{L^2}}{\|u_{exact}\|_{L^2}} \quad (2.46)$$

donde:

u_{pred} : solución aproximada obtenida por la PINN

u_{exact} : solución exacta o numérica de alta fidelidad

$\| \cdot \|_{L^2}$: norma L^2 , que mide la magnitud cuadrática promedio de una función.

3. Ventajas de los PINNs:

- Sin malla: No requieren discretización espacial.
- Flexibles: Pueden manejar geometrías complejas.
- Problemas inversos: Identifican parámetros desconocidos (ej. viscosidad en fluidos).
- Fusión de datos: Combina mediciones escasas con ecuaciones físicas.

4. Desventajas:

- Coste computacional: Requieren muchos recursos para entrenar.
- Inestabilidad: A veces convergen a soluciones erróneas.
- Hiperparámetros sensibles: Elección de arquitectura y optimizador es crucial.

2.3. Ecuación de Schrödinger No Lineal (NLSE)

Descripción de la NLSE y su importancia en física

La NLSE es una ecuación en derivadas parciales no lineal que describe la evolución de paquetes de ondas en medios dispersivos y no lineales. Su forma general es:

$$i\frac{\partial\psi}{\partial t} + \frac{1}{2}\frac{\partial^2\psi}{\partial x^2} + \gamma|\psi|^2\psi = 0 \quad (2.47)$$

Donde:

$\psi(x, t)$: Función de onda compleja.

γ : Parametro de no linealidad el cual cumple que si $\gamma > 0$ (focusing) o $\gamma < 0$ (defocusing) condicionando si se forman solitones brillantes o oscuros.

Importancia:

Modela fenómenos donde la dispersión y la no linealidad se equilibran, permitiendo soluciones localizadas como solitones.

Es fundamental en óptica no lineal, física de materia condensada y dinámica de fluidos.

Soluciones conocidas de la NLSE

Un aspecto clave de la NLSE es su capacidad para modelar tanto solitones, que son paquetes de ondas estables y localizados, como rogue waves, que son ondas excepcionalmente grandes e impredecibles.

- Solitones

Los solitones son paquetes de ondas estables que se autorrefuerzan y mantienen su forma y velocidad durante la propagación, incluso después de colisiones. Surgen de un equilibrio entre la dispersión y la no linealidad en el NLSE.

Ejemplos de solitones incluyen solitones brillantes (pulsos localizados de alta intensidad) y solitones oscuros (regiones de intensidad reducida dentro de un fondo uniforme).

-Solitón fundamental:

$$\psi(x, t) = A \operatorname{sech}(A(x - vt)) e^{i(vx + (A^2 - v^2)t/2)} \quad (2.48)$$

- Rogue waves

Las Rogue wave son olas extremadamente grandes e impredecibles que aparecen repentina e inesperadamente, modelan olas gigantes en océanos y pulsos intensos en láseres. Constituyen un peligro significativo en muchos sistemas físicos, especialmente en el océano.

El NLSE proporciona un marco matemático para comprender la formación de olas gigantes, con soluciones como el solitón peregrino, que describe olas que surgen de la nada. Mecanismos como la inestabilidad modulacional y la colisión de paquetes de olas pueden provocar la generación de olas gigantes. La figura 2.7 representa un soliton peregrino.

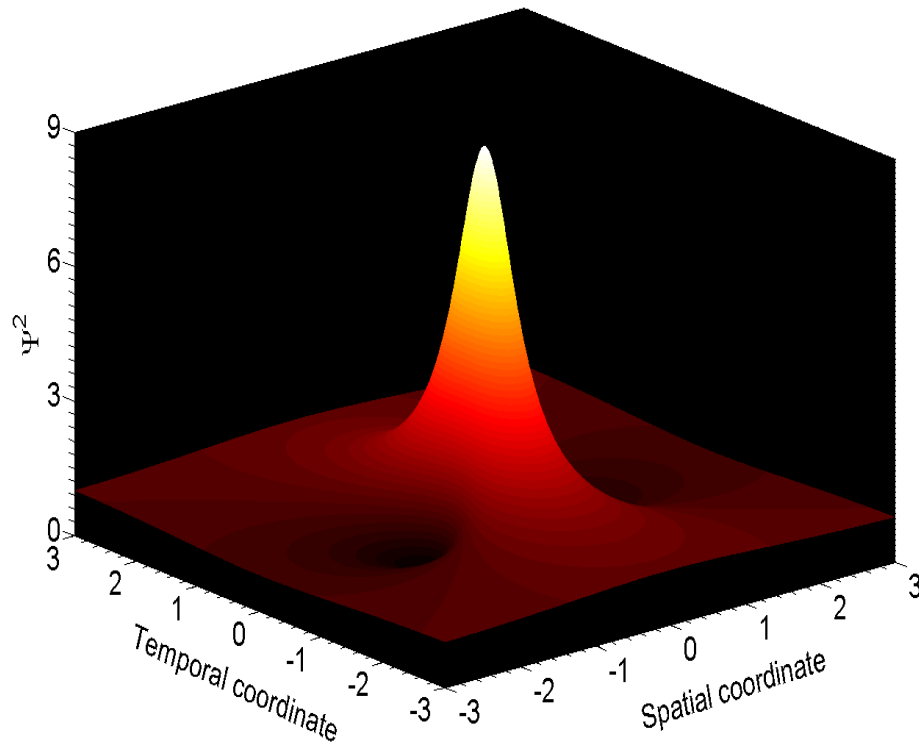


Figura 2.7: Vista 3D de la evolución espacio-temporal de un solitón peregrino. Fuente : https://en.wikipedia.org/wiki/Peregrine_soliton

- Breathers

Un "breather"(oscilón) es un tipo de solitón que exhibe oscilaciones temporales y espaciales, localizado en el espacio y con energía que oscila en el tiempo. Existen respiradores fijos (oscilones) y móviles.

La energía se acumula y se disipa de forma oscilatoria, con amplitudes que varían periódicamente.

Sus tipos son fijos (oscilones), cuya amplitud varía con el tiempo, y móviles.

Aplicaciones de la NLSE en diferentes contextos físicos

- Fibras ópticas

La NLSE describe la propagación de pulsos de luz en fibras, donde:

La dispersión cromática y el efecto Kerr no lineal compiten.

Solitones ópticos permiten transmitir información a largas distancias sin distorsión.

- Condensados de Bose-Einstein(BEC)

La NLSE (como ecuación de Gross-Pitaevskii) modela la dinámica de condensados cuánticos:

$$i\hbar\frac{\partial\psi}{\partial t} = \left(-\frac{\hbar^2}{2m}\nabla^2 + V_{ext} + g|\psi|^2\right)\psi \quad (2.49)$$

aquí, g representa interacciones atómicas que dependiendo su signo se presentan fenomenos repulsivos o atractivos.

- Hidrodinámica y plasmas

La NLSE aparece en la descripción de ondas de superficie en fluidos y ondas de Langmuir en plasmas.

2.4. Aprendizaje profundo basado en la física: Solucion basada en datos de la ecuacion diferencial parcial no lineal de Shrodinger (ref. [2])

A primera vista, la tarea de entrenar un algoritmo de aprendizaje profundo para identificar con precisión un mapa no lineal a partir de unos pocos pares de datos de entrada y salida, potencialmente de muy alta dimensión, parece, en el mejor de los casos, ingenua. En estos casos, existe una gran cantidad de conocimiento previo que actualmente no se utiliza en la práctica moderna del aprendizaje automático. Ya sean las leyes físicas fundamentales que rigen la dinámica dependiente del tiempo de un sistema, o algunas reglas empíricas validadas u otros conocimientos especializados, esta información previa puede actuar como un agente de regularización que limita el espacio de soluciones admisibles a un tamaño manejable (por ejemplo, en problemas de dinámica de fluidos incompresibles, descartando cualquier solución de flujo no realista que viole el principio de conservación de la masa). A cambio, codificar dicha información estructurada en un algoritmo de aprendizaje amplifica el contenido informativo de los datos que el algoritmo ve, lo que le permite orientarse rápidamente hacia la solución correcta y generalizar eficazmente incluso cuando solo se dispone de unos pocos ejemplos de entrenamiento.

Este ejemplo resalta la capacidad del método para manejar condiciones de contorno periódicas, soluciones de valores complejos, así como diferentes tipos de no linealidades en las ecuaciones diferenciales parciales que las rigen. La ecuación unidimensional no lineal de Schrodinger es una ecuación de campo clásica que se utiliza para estudiar sistemas mecánicos cuánticos, incluyendo la propagación de ondas no lineales en fibras ópticas y/o guías de onda, condensados de Bose-Einstein y ondas de plasma. En óptica, el término no lineal surge del índice de refracción dependiente de la intensidad de un material dado. De manera similar, el término no lineal para los condensados de Bose-Einstein es el resultado de las interacciones del campo medio de un sistema de N cuerpos en interacción. La ecuación no lineal de Schrödinger, junto con las condiciones de contorno periódicas, viene dada por

$$ih_t + \frac{1}{2}h_{xx} + |h|^2h = 0, \quad x \in [-5, 5], \quad t \in [0, \pi/2]$$

$$\begin{aligned} h(0, x) &= 2\operatorname{sech}(x), \\ h(t, -5) &= h(t, 5), \\ h_x(t, -5) &= h_x(t, 5) \end{aligned} \tag{2.50}$$

donde $h(t, x)$ es la solución compleja. Definiendo $f(t, x)$ como

$$f := ih_t + \frac{1}{2}h_{xx} + |h|^2h,$$

Por lo que se coloca una red neuronal de valor complejo a en $h(t, x)$. De hecho, si u denota la parte real de h y v es la parte imaginaria, se coloca la red neuronal de múltiples salidas en $h(t, x) = [u(t, x)v(t, x)]$. Esto dará como resultado la red neuronal $f(t, x)$ con información física y valores complejos (múltiples salidas). Los parámetros compartidos de las redes neuronales $h(t, x)$ y $f(t, x)$ pueden aprenderse minimizando la pérdida de error cuadrático medio.

$$MSE = MSE_0 + MSE_b + MSE_f, \tag{2.51}$$

donde

$$MSE_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |h(0, x_0^i) - h_0^i|^2,$$

$$MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} (|h^i(t_b^i, -5) - h^i(t_b^i, 5)|^2 + |h_x^i(t_b^i, -5) - h_x^i(t_b^i, 5)|^2),$$

y

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

Aquí, $\{x_0^i, h_0^i\}_{i=1}^{N_0}$ denota los datos iniciales, $\{t_b^i\}_{i=1}^{N_b}$ corresponden a los puntos de colocación en el contorno, y $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ representan los puntos de colocación en $f(t, x)$. En consecuencia, MSE_0 corresponde a la pérdida en los datos iniciales, MSE_b impone las condiciones de contorno periódicas y MSE_f penaliza el incumplimiento de la ecuación de Schrödinger en los puntos de colocación.

Para evaluar la precisión del método, se simula la ecuación (2.46) utilizando métodos espectrales convencionales para crear un conjunto de datos

de alta resolución. En concreto, a partir de un estado inicial $h(0, x) = 2\text{sech}(x)$ y asumiendo condiciones periódicas de contorno $h(t, -5) = h(t, 5)$ y $h_x(t, -5) = h_x(t, 5)$, se integro la ecuación (2.46) hasta un tiempo final $t = \pi/2$ utilizando el paquete Chebfun con una discretización espectral de Fourier con 256 modos y un integrador temporal de Runge-Kutta explícito de cuarto orden con paso de tiempo $\Delta t = \pi/2 \cdot 10^{-6}$. En el entorno basado en datos, todo lo que se observan son mediciones $\{x_0^i, h_0^i\}_{i=1}^{N_0}$ de la función latente $h(t, x)$ en tiempo $t = 0$. En particular, el conjunto de entrenamiento consta de un total de $N_0 = 50$ puntos de datos en $h(0, x)$ analizados aleatoriamente del conjunto completo de datos de alta resolución, así como $N_b = 50$ puntos de colocación muestreados aleatoriamente $\{t_b^i\}_{i=1}^{N_b}$ para aplicar los límites periódicos. Además, se asumen $N_f = 20000$ puntos de colocación muestreados aleatoriamente para aplicar la ecuación (2.46) dentro del dominio de la solución. Todas las ubicaciones de los puntos muestreados aleatoriamente se generaron mediante una estrategia de muestreo Latin Hypercube que llena el espacio.

El objetivo aquí es inferir la solución espacio-temporal completa $h(t, x)$ de la ecuación de Schrödinger (2.46). Optando por representar conjuntamente la función latente $h(t, x) = [u(t, x)v(t, x)]$ usando una red neuronal profunda de 5 capas con 100 neuronas por capa y una función de activación tangente hiperbólica. La Figura 2.7 resume los resultados del experimento. En concreto, el panel superior de la Figura 2.7 muestra la magnitud de la solución espacio-temporal prevista $|h(t, x)| = \sqrt{u^2(t, x) + v^2(t, x)}$ junto con las ubicaciones de los datos de entrenamiento iniciales y de límite. El error de predicción resultante se valida con los datos de prueba para este problema y se mide en $8,49 \cdot 10^{-3}$ en la norma L_2 relativa. En el panel inferior de la Figura 2.7 se presenta una evaluación más detallada de la solución predicha. En particular, se presenta una comparación entre las soluciones exactas y predichas en diferentes instantes de tiempo: $t = 0.20, 0.79, 1.18$. Utilizando solo unos pocos datos iniciales, la red neuronal basada en la física puede capturar con precisión el complejo comportamiento no lineal de la ecuación de Schrödinger.

Una posible limitación de los modelos de redes neuronales de tiempo continuo reside en la necesidad de utilizar un gran número de puntos de colocación N_f para aplicar restricciones basadas en la física en todo el dominio espaciotemporal. Si bien esto no supone problemas significativos para pro-

blemas de una o dos dimensiones espaciales, puede introducir un cuello de botella importante en problemas de dimensiones superiores, ya que el número total de puntos de colocación necesarios para aplicar globalmente una restricción basada en la física aumentará exponencialmente.

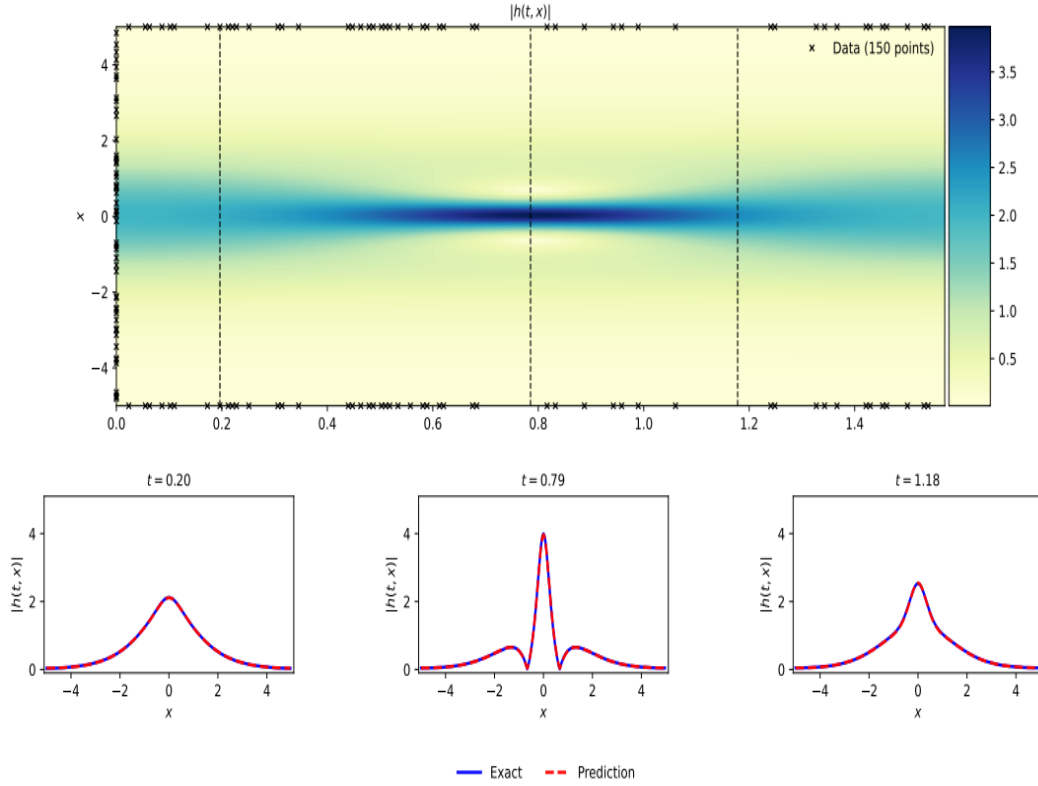


Figura 2.8: Arriba: Solución predicha $|h(t, x)|$ junto con los datos de entrenamiento iniciales y de límite. Utilizando 20000 puntos de colocación generados mediante una estrategia de muestreo Latin Hypercube. Abajo: Comparación de las soluciones predichas y exactas correspondientes a las tres instantáneas temporales representadas por las líneas verticales discontinuas en el panel superior. El error L_2 relativo para este caso es $8,49 \cdot 10^{-3}$

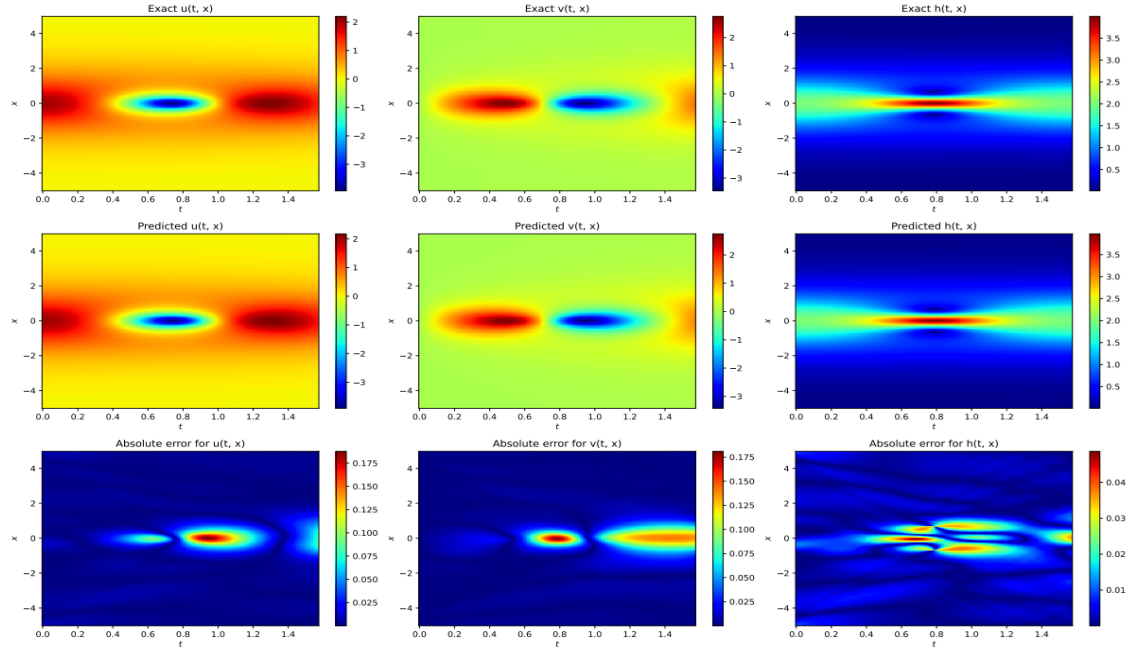


Figura 2.9: Las imágenes del primer renglon son gráficos que muestran las soluciones exactas para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Los ejes representan x y t , con valores en una región específica. Las funciones presentan estructuras bien definidas en el espacio-tiempo.

Las imágenes del segundo renglon son gráficos que muestran las predicciones obtenidas mediante la red neuronal para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Visualmente, las predicciones parecen bastante similares a las soluciones exactas.

Las imágenes del tercer renglon son gráficos adicionales que muestran el error absoluto entre la solución exacta y la predicha para cada función. Los errores son relativamente pequeños, con valores máximos cercanos a 0.175 para $u(x, t)$ y $v(x, t)$, y menores para $h(x, t)$.

En general, la imagen sugiere que la red neuronal ha aprendido bastante bien la solución a la ecuación, con errores relativamente pequeños en su predicción.

- Solución tipo soliton

Una solución tipo soliton de la ecuación no lineal de Schrodinger de la forma $q(x, t) = 2\eta \text{sech}[2\eta(x + \kappa t - x_0)] \exp\{-2i[\kappa x + 2(\kappa^2 - \eta^2)t + \phi_0]\}$ con parámetros $\eta = 1$, $\kappa = 1$, $x_0 = 0$ y $\phi_0 = 1$ se escribe como $q(x, t) =$

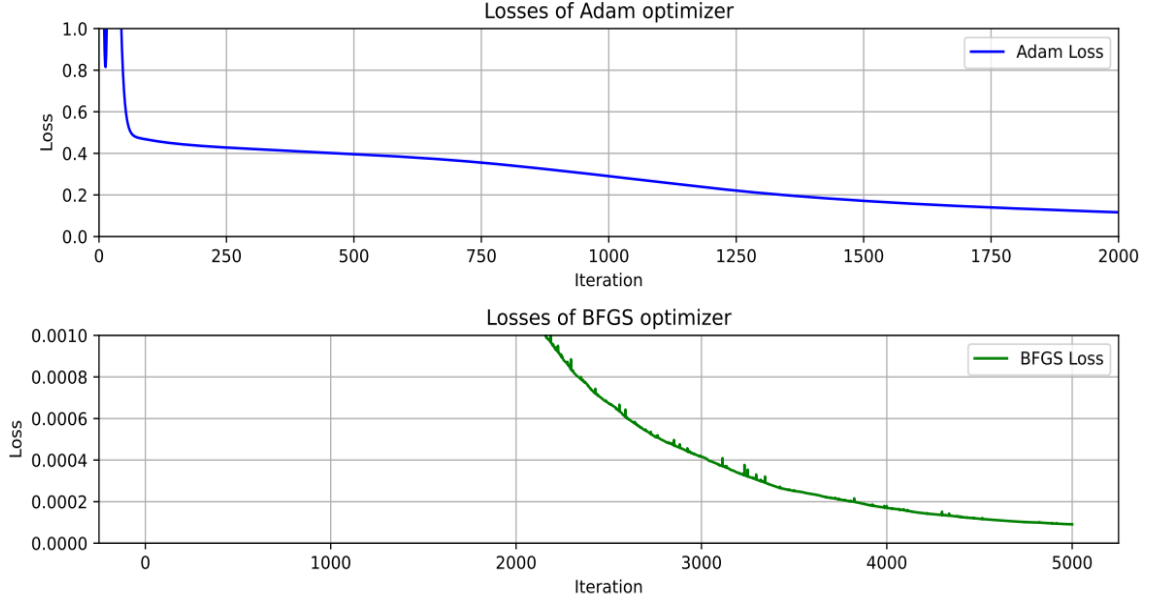


Figura 2.10: La red neuronal es entrenada con los optimizadores Adam y L-BFGS-B. La figura de arriba muestra el descenso de la funcion Loss para el optimizador Adam el cual realizo 2000 iteraciones, la imagen de abajo muestra el descenso para el optimizador L-BFGS-B que realiza 5000 iteraciones.

La PINN realizo este aprendizaje en un tiempo de 3693s (aprox. 61.5min) en un cpu i5-12400 para el problema (2.46)

$2\text{sech}(8t + 2x)\exp(-2ix + i)$ (ref. [12]) y se usa para crear un conjunto de datos con los valores de frontera $x \in [-6, 6]$ en el tiempo $t \in [0, 1]$ figura 2.11

El conjunto de entrenamiento consta de un total de $N_0 = 100$ puntos de datos en $h(0, x)$ analizados aleatoriamente del conjunto completo de datos de alta resolución, así como $N_b = 200$ puntos de colocación muestreados aleatoriamente $\{t_b^i\}_{i=1}^{N_b}$ para aplicar los límites periódicos. Además, se asumen $N_f = 20000$ puntos de colocación muestreados aleatoriamente para aplicar la ecuación (2.46) dentro del dominio de la solución de la figura 2.11.

Usando una red neuronal profunda de 5 capas con 100 neuronas por capa y una función de activacion tangente hiperbólica. La Figura 2.12 resume los

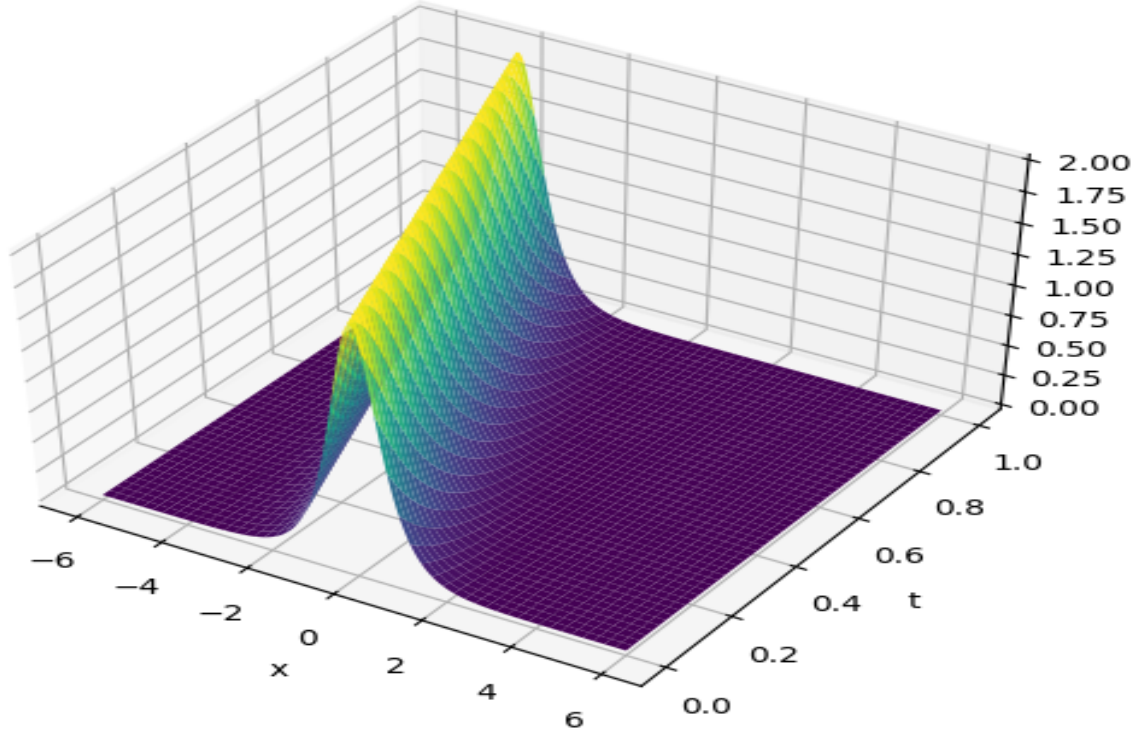


Figura 2.11: Grafica del soliton $h(x, t) = 2\text{sech}(8t + 2x)\exp(-2ix + i)$ de la que se toman los datos que se muestrean utilizando un muestreo Latin hypercube para entrenar la red neuronal.

resultados del experimento. En concreto, el panel superior de la Figura 2.12 muestra la magnitud de la solución espacio-temporal prevista $|h(t, x)| = \sqrt{u^2(t, x) + v^2(t, x)}$ junto con las ubicaciones de los datos de entrenamiento iniciales y de límite. El error de predicción resultante se valida con los datos de prueba para este problema y se mide en $2,47 \cdot 10^{-2}$ en la norma L_2 relativa. En el panel inferior de la Figura 2.12 se presenta una evaluación más detallada de la solución predicha. En particular, se presenta una comparación entre las soluciones exactas y predichas en diferentes instantes de tiempo: $t = .25, .5, .75$. Utilizando solo unos pocos datos iniciales, la red neuronal basada en la física puede capturar con precisión el complejo comportamiento no lineal de la ecuación no lineal de Schrodinger con la solución tipo soliton.

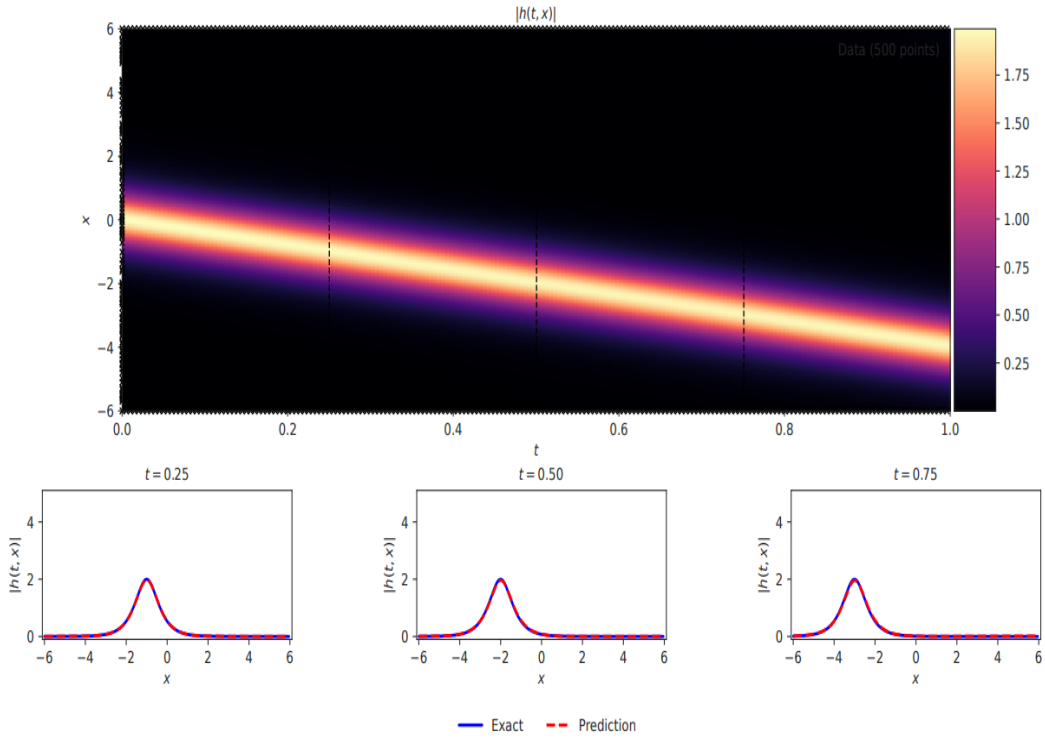


Figura 2.12: Arriba: Solución predicha $|h(t, x)|$ junto con los datos de entrenamiento iniciales y de límite. Utilizando 20000 puntos de colocación generados mediante una estrategia de muestreo Latin Hypercube. Abajo: Comparación de las soluciones predichas y exactas correspondientes a las tres instantáneas temporales representadas por las líneas verticales discontinuas en el panel superior. El error L_2 relativo para este caso es $2,47 \cdot 10^{-2}$

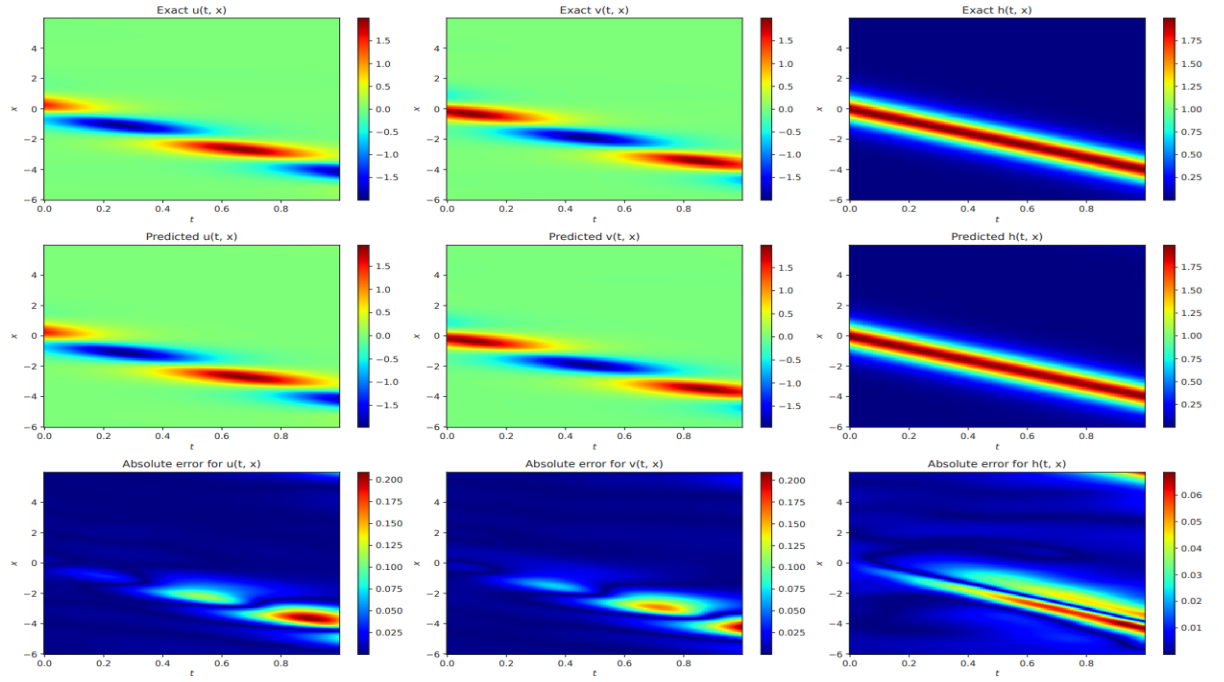


Figura 2.13: Las imágenes del primer renglón son gráficos que muestran las soluciones exactas para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Los ejes representan x y t , con valores en una región específica. Las funciones presentan estructuras bien definidas en el espacio-tiempo.

Las imágenes del segundo renglón son gráficos que muestran las predicciones obtenidas mediante la red neuronal para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Visualmente, las predicciones parecen bastante similares a las soluciones exactas.

Las imágenes del tercer renglón son gráficos adicionales que muestran el error absoluto entre la solución exacta y la predicha para cada función. Los errores son relativamente pequeños, con valores no mayores a 0.2 para $u(x, t)$ y $v(x, t)$, y menores para $h(x, t)$.

En general, la imagen sugiere que la red neuronal ha aprendido bastante bien la solución a la ecuación, con errores relativamente pequeños en su predicción.

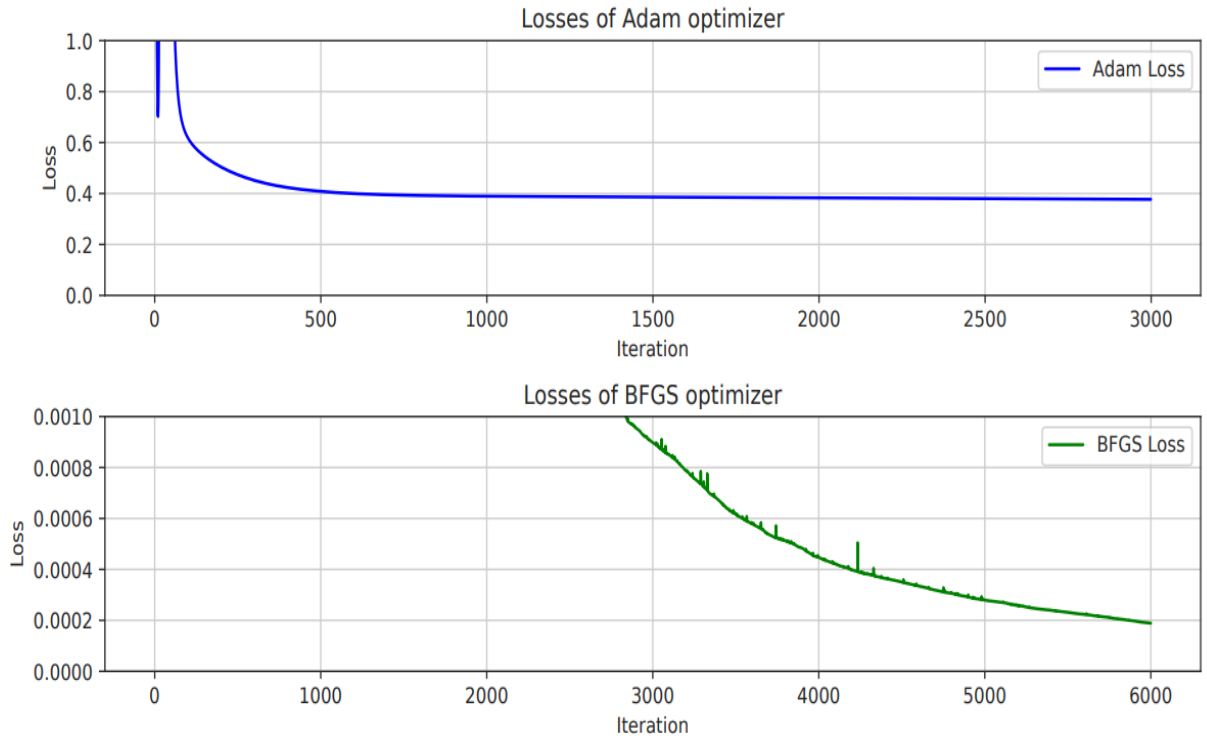


Figura 2.14: La red neuronal es entrenada con los optimizadores Adam y L-BFGS-B. La figura de arriba muestra el descenso de la función Loss para el optimizador Adam el cual realizó 3000 iteraciones, la imagen de abajo muestra el descenso para el optimizador L-BFGS-B que realizó 6000 iteraciones.

La PINN realizó este aprendizaje en un tiempo de 5486s (aprox. 1.5hrs.) en un cpu i5-12400

Aprendizaje profundo basado en la física: Solución basada en datos de la ecuación no lineal de Hirota

La ecuación unidimensional no lineal de Hirota es una ecuación de evolución integrable que describe la propagación de ondas en medios no lineales dispersivos.

Esta ecuación es una versión modificada de la ecuación de Schrödinger no lineal con términos de dispersión y no linealidad asimétrica, y es integrable

en el sentido de la teoría de solitones. La ecuación no lineal de Hirota, viene dada por

$$ih_t + \alpha(h_{xx} + 2|h|^2h) + i\beta(h_{xxx} + 6|h|^2h_x) = 0 \quad (2.52)$$

donde $q(t, x)$ es la solución compleja y se consideran los valores $\alpha = 1$ $\beta = ,01$. Definiendo $f(t, x)$ de una forma analoga al caso del ejemplo de la NLSE

Por lo que se coloca una red neuronal de valor complejo a en $h(t, x)$. De hecho, si u denota la parte real de h y v es la parte imaginaria, se coloca la red neuronal de múltiples salidas en $q(t, x) = [u(t, x)v(t, x)]$. Esto dará como resultado la red neuronal $f(t, x)$ con información física y valores complejos (múltiples salidas). Los parámetros compartidos de las redes neuronales $h(t, x)$ y $f(t, x)$ pueden aprenderse minimizando la pérdida de error cuadrático medio (2.47).

- Solución tipo soliton brillante

Sea la solución tipo soliton brillante de la ecuación de Hirota (ref. [10]) de la forma $h(t, x) = \text{sech}(x - \beta t)\exp(it)$ donde $\beta = 0,01$ para crear un conjunto de datos con los valores de frontera $x \in [-10, 10]$ en el tiempo $t \in [0, 5]$ figura 2.15 .

El conjunto de entrenamiento consta de un total de $N_0 = 100$ puntos de datos en $h(0, x)$ analizados aleatoriamente del conjunto completo de datos de alta resolución, así como $N_b = 100$ puntos de colocación muestreados aleatoriamente $\{t_b^i\}_{i=1}^{N_b}$ para aplicar los límites periódicos. Además, se asumen $N_f = 10000$ puntos de colocación muestreados aleatoriamente para aplicar la ecuación (2.48) dentro del dominio de la solución.

Usando una red neuronal profunda de 5 capas con 100 neuronas por capa y una función de activación tangente hiperbólica. La Figura 2.16 resume los resultados del experimento. En concreto, el panel superior de la Figura 2.16 muestra la magnitud de la solución espacio-temporal prevista $|h(t, x)| = \sqrt{u^2(t, x) + v^2(t, x)}$ junto con las ubicaciones de los datos de entrenamiento iniciales y de límite. El error de predicción resultante se valida con los datos de prueba para este problema y se mide en $9,7 \cdot 10^{-3}$ en la norma L_2 relativa. En el panel inferior de la Figura 2.16 se presenta una evaluación más detallada de la solución predicha. En particular, se presenta una comparación

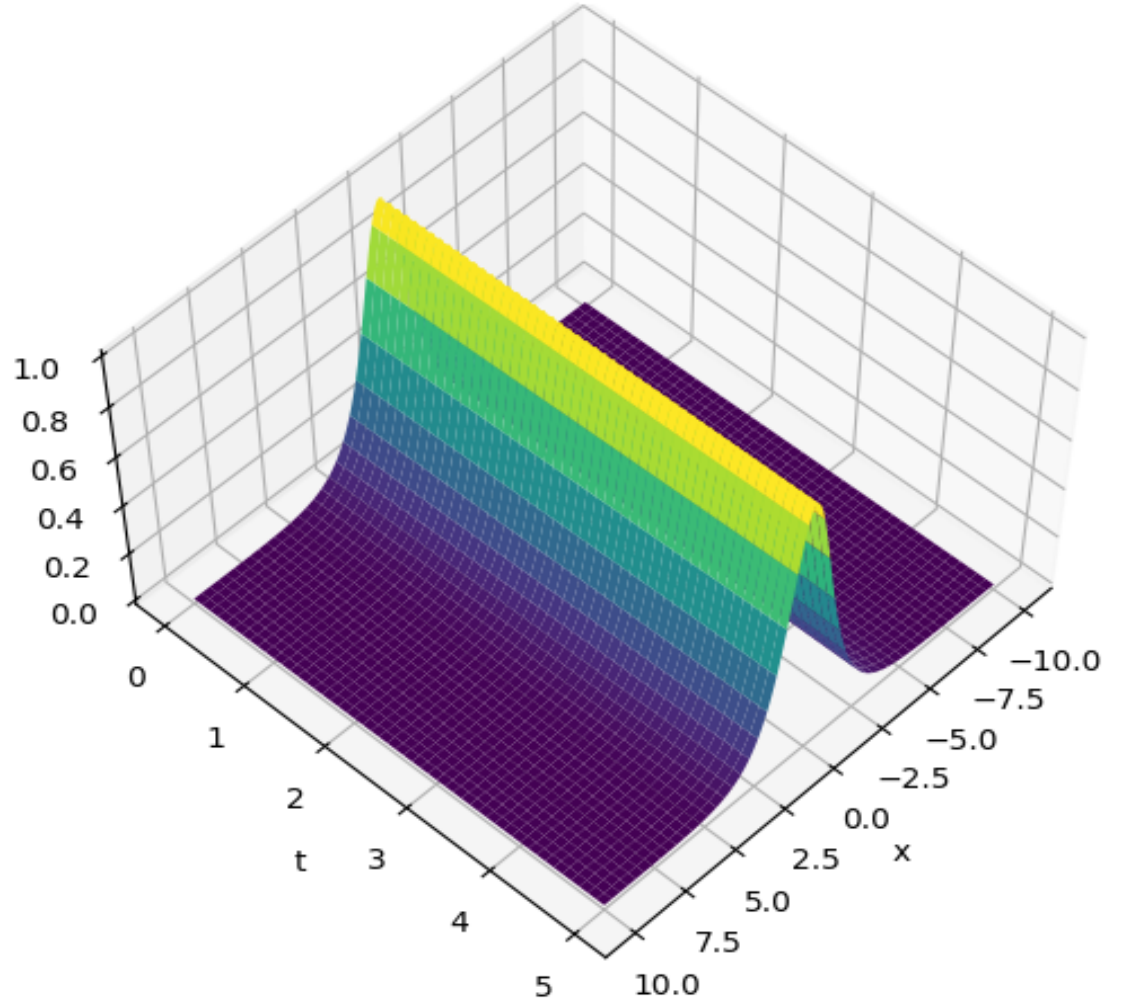


Figura 2.15: Grafica del soliton brillante $h(t, x) = \text{sech}(x - \beta t) \exp(it)$ donde $\beta = 0,01$ de la que se toman los datos que se muestrean utilizando un muestreo Latin hypercube para entrenar la red neuronal.

entre las soluciones exactas y predichas en diferentes instantes de tiempo: $t = 1.25, 2.5, 3.75$. Utilizando solo unos pocos datos iniciales, la red neuronal basada en la física puede capturar con precisión el complejo comportamiento no lineal de la ecuación de Hirota con la solución tipo soliton brillante.

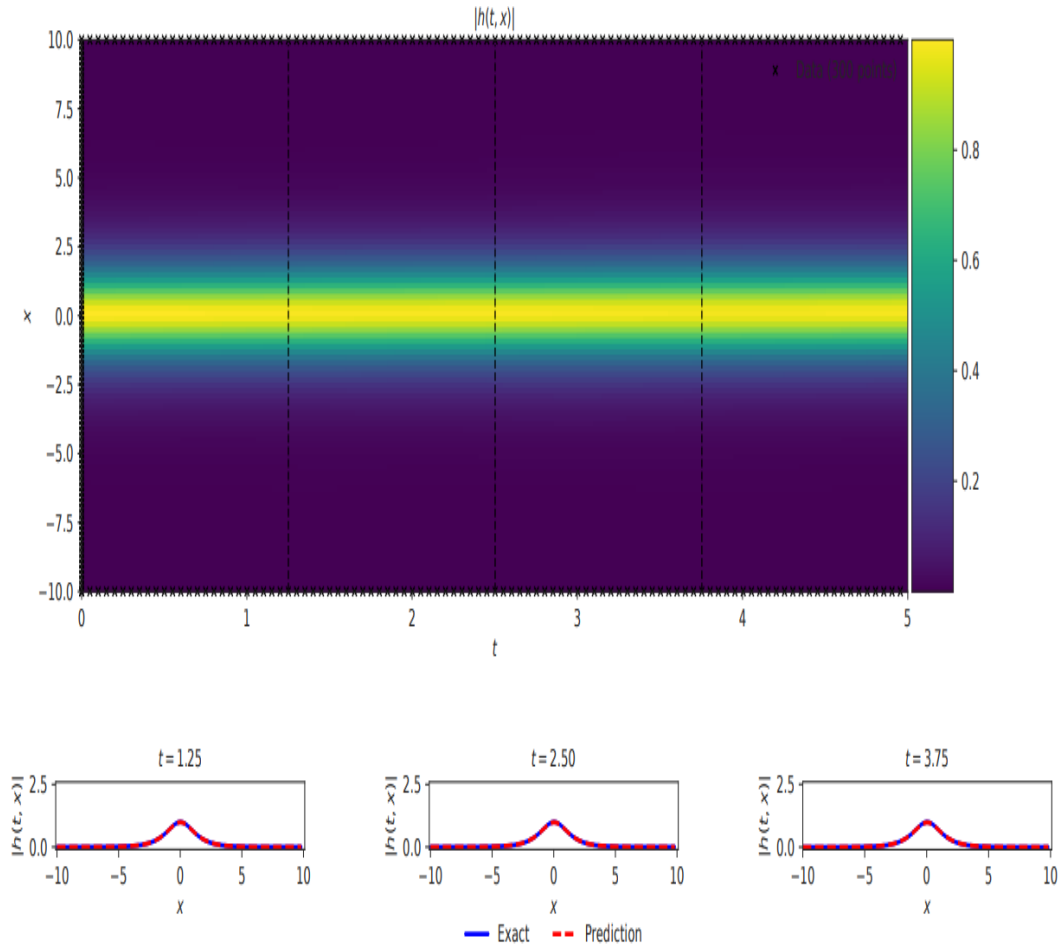


Figura 2.16: Arriba: Solución predicha $|h(t, x)|$ junto con los datos de entrenamiento iniciales y de límite. Utilizando 10000 puntos de colocación generados mediante una estrategia de muestreo Latin Hypercube. Abajo: Comparación de las soluciones predichas y exactas correspondientes a las tres instantáneas temporales representadas por las líneas verticales discontinuas en el panel superior. El error L_2 relativo para este caso es $9,7 \cdot 10^{-3}$

- Solución AKM breather

El AKM breather

$$h(x, t) = \frac{\cosh(\omega t - 2ic) - \cos(c)\cos(p\xi)}{(\cosh(\omega t) - \cos(c)\cos(p\xi))\exp(2it)} \quad (2.53)$$

donde

$$\begin{aligned} \xi &= x - 2\beta(2 + \cos(2c)t) \\ \omega &= 2\sin(2c) \\ p &= 2\sin(c) \end{aligned} \quad (2.54)$$

con $\beta = 1$, $c = .75$ y $\alpha = 1$ para crear un conjunto de datos con los valores de frontera $x \in [-10, 10]$ en el tiempo $t \in [0, 1]$ figura 2.19

El conjunto de entrenamiento consta de un total de $N_0 = 200$ puntos de datos en $h(0, x)$ analizados aleatoriamente del conjunto completo de datos de alta resolución, así como $N_b = 200$ puntos de colocación muestreados aleatoriamente $\{t_b^i\}_{i=1}^{N_b}$ para aplicar los límites periódicos. Además, se asumen $N_f = 20000$ puntos de colocación muestreados aleatoriamente para aplicar la ecuación (2.48) dentro del dominio de la solución.

Usando una red neuronal profunda de 5 capas con 100 neuronas por capa y una función de activación tangente hiperbólica. La Figura 2.20 resume los resultados del experimento. En concreto, el panel superior de la Figura 2.20 muestra la magnitud de la solución espacio-temporal prevista $|h(t, x)| = \sqrt{u^2(t, x) + v^2(t, x)}$ junto con las ubicaciones de los datos de entrenamiento iniciales y de límite. El error de predicción resultante se valida con los datos de prueba para este problema y se mide en $2,93 \cdot 10^{-1}$ en la norma L_2 relativa. En el panel inferior de la Figura 2.20 se presenta una evaluación más detallada de la solución predicha. En particular, se presenta una comparación entre las soluciones exactas y predichas en diferentes instantes de tiempo: $t = .25, .50, .75$. Los resultados no muestran resultados de error de un orden aceptable esto puede ser debido a los siguientes causas:

Causas Principales

-Complejidad de la Solución AKM Breather:

Las soluciones tipo breather son altamente no lineales y oscilatorias, lo que

las hace particularmente difíciles de aproximar con redes neuronales.

-Arquitectura de la Red:

El usar una red con 4 capas ocultas de 100 neuronas cada una, puede ser insuficiente para esta complejidad.

La función de activación es tanh en todas las capas, lo que podría no ser la más adecuada para este tipo de problema.

-Pérdida de Optimización:

Las gráficas de pérdida muestran que el optimizador Adam no reduce suficientemente la pérdida antes de cambiar a L-BFGS-B.

-Puntos de Entrenamiento:

El usar 200 puntos para las condiciones iniciales (N_0) y de frontera (N_b), lo que podría ser insuficiente.

Los 20,000 puntos colocalizados internos (N_f) podrían no estar bien distribuidos para capturar la dinámica compleja.

-Hiperparámetros:

Los parámetros $\alpha = 1$ y $\beta=0.01$ en la ecuación podrían necesitar ajuste.

El uso de normalización de los datos más allá del escalado inicial podría ayudar a mejorar el aprendizaje.

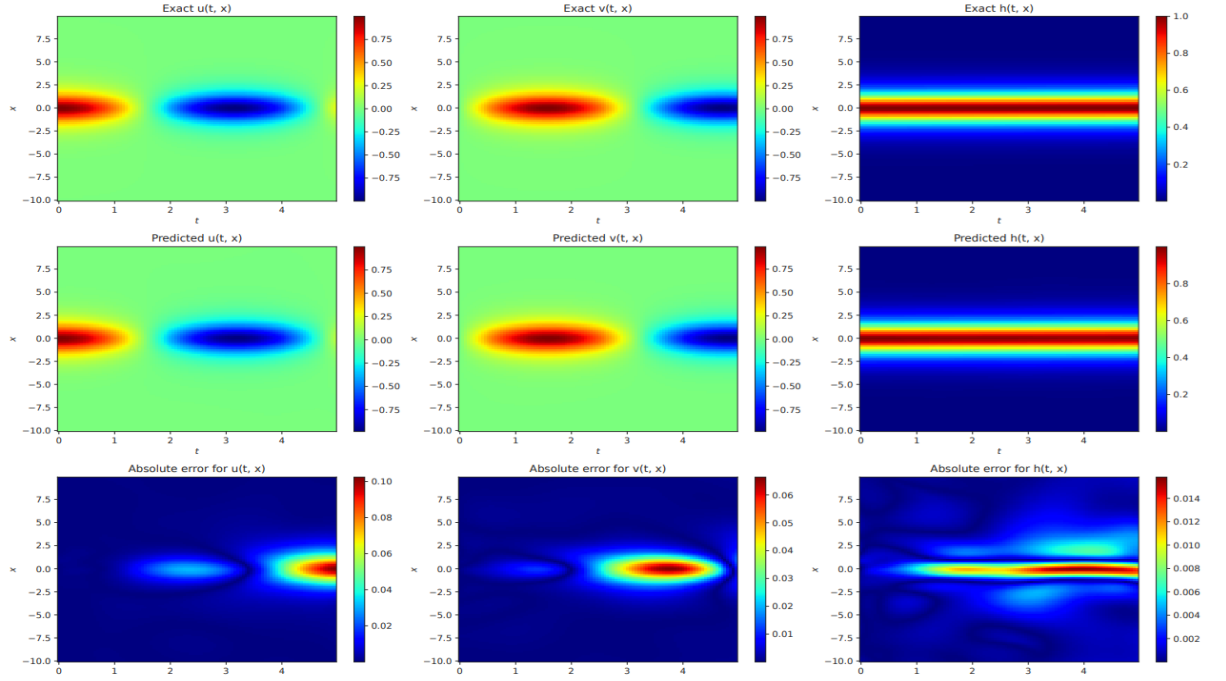


Figura 2.17: Las imágenes del primer renglón son gráficos que muestran las soluciones exactas para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Los ejes representan x y t , con valores en una región específica. Las funciones presentan estructuras bien definidas en el espacio-tiempo.

Las imágenes del segundo renglón son gráficos que muestran las predicciones obtenidas mediante la red neuronal para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Visualmente, las predicciones parecen bastante similares a las soluciones exactas.

Las imágenes del tercer renglón son gráficos adicionales que muestran el error absoluto entre la solución exacta y la predicha para cada función. Los errores son relativamente pequeños, con valores no mayores a 0.1 $u(x, t)$ y $v(x, t)$, y menores para $h(x, t)$.

En general, la imagen sugiere que la red neuronal ha aprendido bastante bien la solución a la ecuación, con errores relativamente pequeños en su predicción.

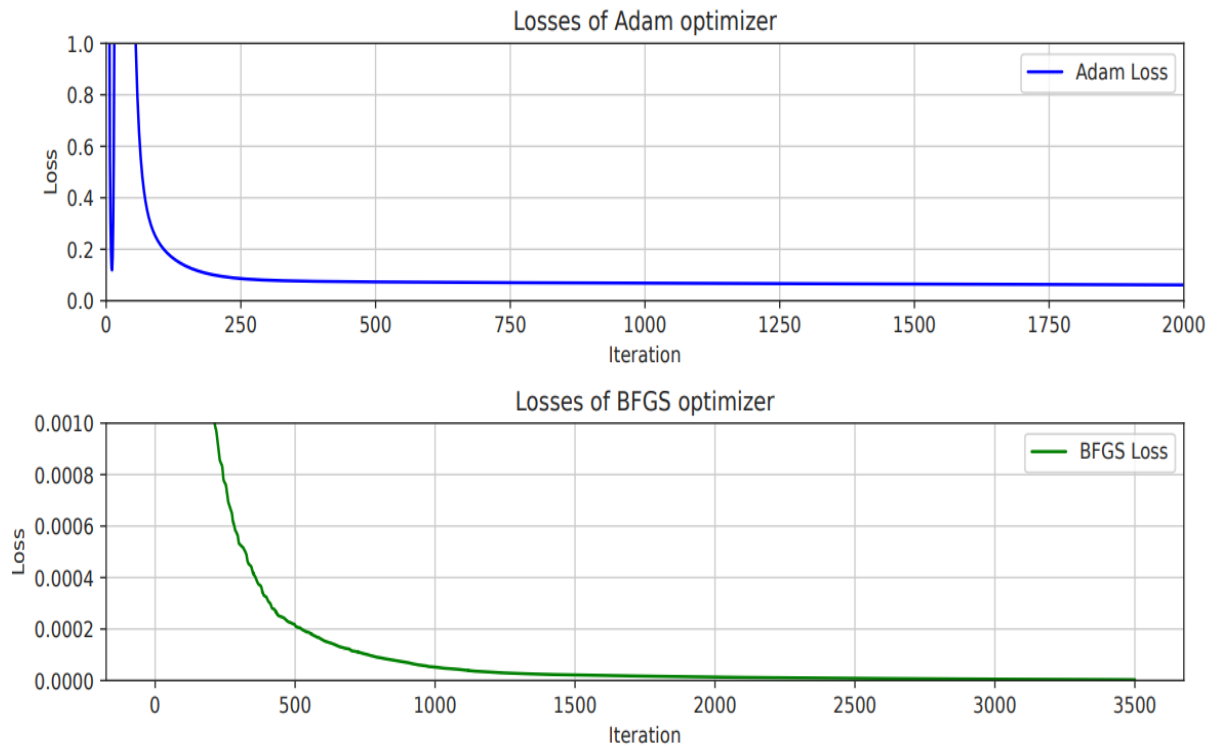


Figura 2.18: La red neuronal es entrenada con los optimizadores Adam y L-BFGS-B. La figura de arriba muestra el descenso de la funcion Loss para el optimizador Adam el cual realizo 2000 iteraciones, la imagen de abajo muestra el descenso para el optimizador L-BFGS-B que realiza 3500 iteraciones.

La PINN realizo este aprendizaje en un tiempo de 3514s (aprox. 58.5min) en un cpu i5-12400

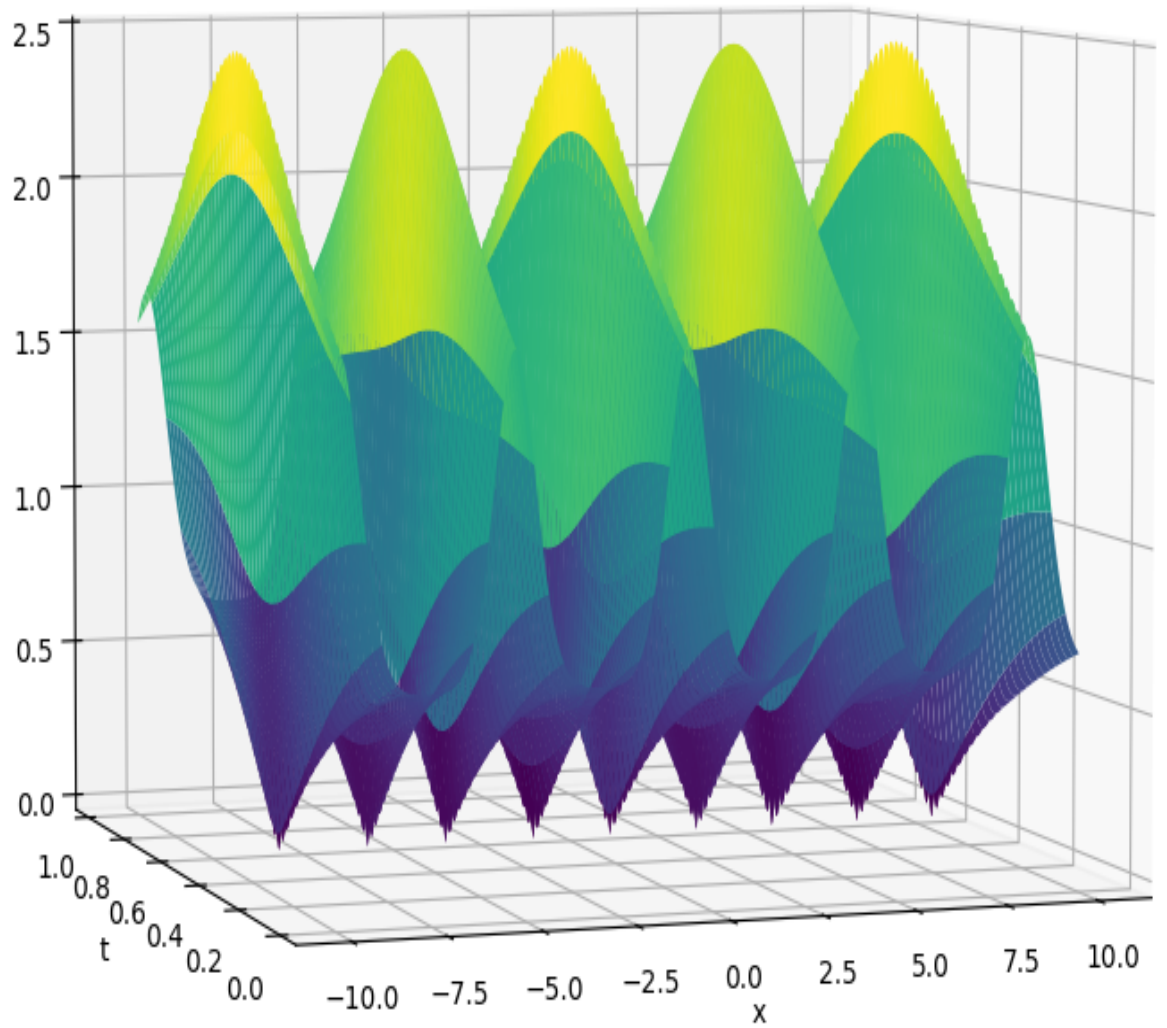


Figura 2.19: Grafica del AKM breather $h(x, t) = \frac{\cosh(\omega t - 2ic) - \cos(c)\cos(p\xi)}{(\cosh(\omega t) - \cos(c)\cos(p\xi))\exp(2it)}$ de la que se toman los datos que se muestrean utilizando un muestreo Latin hypercube para entrenar la red neuronal.

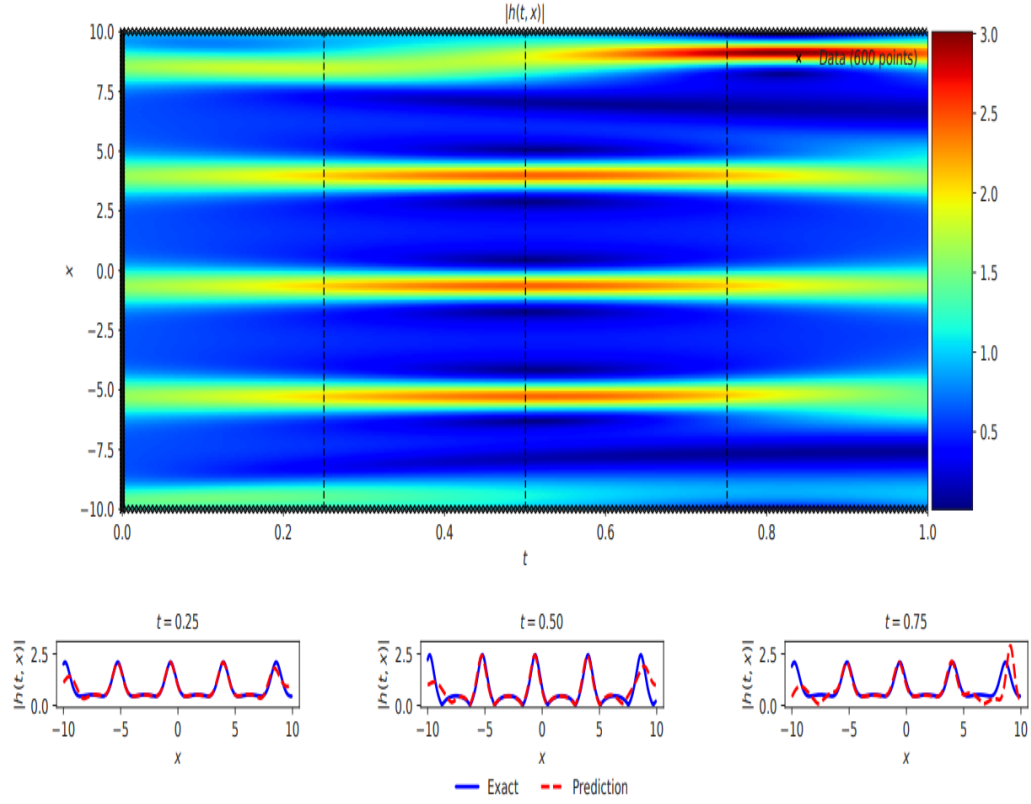


Figura 2.20: Arriba: Solución predicha $|h(t, x)|$ junto con los datos de entrenamiento iniciales y de límite. Utilizando 20000 puntos de colocación generados mediante una estrategia de muestreo Latin Hypercube. Abajo: Comparación de las soluciones predichas y exactas correspondientes a las tres instantáneas temporales representadas por las líneas verticales discontinuas en el panel superior. El error L_2 relativo para este caso es $2,93 \cdot 10^{-1}$

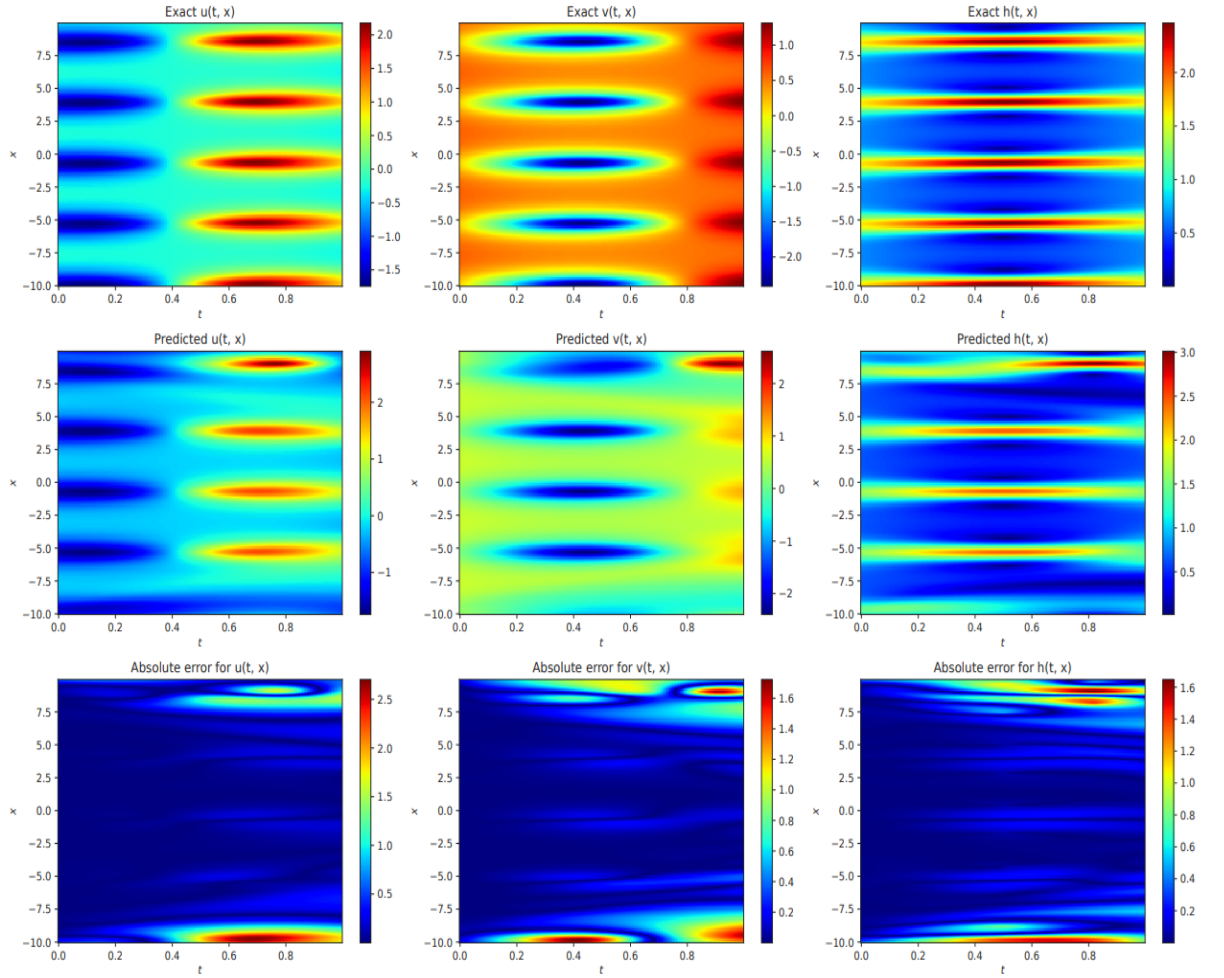


Figura 2.21: Las imágenes del primer renglon son gráficos que muestran las soluciones exactas para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Los ejes representan x y t , con valores en una región específica. Las funciones presentan estructuras bien definidas en el espacio-tiempo.

Las imágenes del segundo renglon son gráficos que muestran las predicciones obtenidas mediante la red neuronal para $u(x, t)$, $v(x, t)$ y $h(x, t)$. Las imágenes del tercer renglon son gráficos adicionales que muestran el error absoluto entre la solución exacta y la predicha para cada función, los errores absolutos en las gráficas muestran patrones estructurados, lo que sugiere que la red está teniendo problemas para capturar ciertas características dinámicas de la solución. Los errores no son lo suficientemente pequeños lo que significa no se puede considerar un aprendizaje deseado.

Capítulo 3

Metodología

3.1. Formulación matemática de la NLSE con potenciales externos.

El concepto clásico de solitón fue desarrollado para sistemas no lineales y dispersivos que han sido autónomos; es decir, el tiempo sólo ha jugado el papel de la variable independiente y no ha aparecido explícitamente en una ecuación no lineal de evolución. Sin embargo, existen situaciones frecuentes cuando un sistema está sujeto a alguna forma de fuerza externa, y estos casos son típicos para experimentos con solitones ópticos temporales y espaciales, láseres de solitones, interruptores y lógica de solitones ultra rápidos (Hasegawa y Kodama, 1995; Agrawal, 2001; Agüero Granados y Serkin, 2020 y Taylor, 1992).

Históricamente, el primer estudio del movimiento acelerado de un solitón en plasma no homogéneo bajo un potencial lineal fue realizado en (ref. [40]) descubrieron que, para el modelo de ecuación de Schrödinger no lineal con un potencial externo lineal, el método de transformación inversa (IST) se puede generalizar permitiendo el valor propio variable en tiempo (TVE). Como consecuencia de esto, los solitones con velocidades que varían en el tiempo (pero con amplitudes invariantes el tiempo) han sido predichos (Chen y Liu, 1976). Este estudio amplió sustancialmente el concepto de solitones clásicos.

La interacción de solitones para los modelos generalizados de la NLSE con

la dispersión, no linealidad, disipación o ganancia y los potenciales variables en tiempo se ha estudiado en trabajos (Serkin y Hasegawa, 2000; Serkin, Hasegawa y Belyaeva, 2007; Serkin, Hasegawa y Belyaeva, 2010; Belyaeva y Serkin, 2011 y T.L. Belyaeva, M.A. Agüero, V.N. Serkin, 2021). Al comportarse como una partícula, un solitón puede acelerarse en un potencial externo cambiando su velocidad, amplitud y fase. Las nuevas clases de solitones, nombrados solitones no autónomos, los solitones no autónomos amplían el concepto clásico al incluir la variación de parámetros como la velocidad, amplitud y fase debido a fuerzas externas. Siendo las soluciones exactas de una ecuación no lineal, solitones no autónomos, interactúan elásticamente, pero mueven con amplitudes, velocidades y espectros variables y adaptados tanto al potencial externo y las variaciones de dispersión y no linealidad.

Uno de los avances más notables en la teoría de ondas no lineales fue el descubrimiento y desarrollo de un método exacto para integrar las ecuaciones no lineales de evolución, conocido como Transformación de Dispersión Inversa (IST). Gardner, Green, Kruskal y Miura demostraron que las soluciones de la ecuación no lineal de Korteweg-de Vries (KdV) presentan un comportamiento de tipo solitón. La cuestión principal es cómo determinar si una ecuación no lineal de evolución es integrable. Para abordar esto, Gardner, Greene, Kruskal y Miura (1967), así como Lax (1968) y el grupo AKNS formado por Ablowitz, Kaup, Newell y Segur (1973), formularon un principio general asociando varias ecuaciones de evolución no lineales con operadores lineales. Este principio fue elegantemente formalizado por Ablowitz, Kaup, Newell y Segur (1973) en lo que se conoce como la jerarquía AKNS.

Existen numerosas ecuaciones de evolución no lineales que tienen soluciones solitónicas, como la ecuación de Korteweg-de Vries, la ecuación de Boussinesq, la ecuación no lineal de Schrödinger, y la ecuación sine-Gordon, entre otras. Cada una de estas ecuaciones puede asociarse con un par de operadores matriciales diferenciales conocidos como pares de Lax, y es posible desarrollar el esquema AKNS para probar su integración exacta. Si una ecuación no lineal de evolución posee pares de Lax, se pueden encontrar sus soluciones solitónicas mediante diversos métodos, como la IST, las transformaciones de Bäcklund, y el método de Hirota (Ablowitz y Segur, 1981).

En este trabajo, aplicamos el método AKNS para demostrar la integrabilidad del sistema, y empleamos la transformada de Bäcklund para obtener soluciones solitónicas. A continuación, describimos ambos enfoques en detalle. Presentando una investigación original enfocada en el análisis de los solitones no autónomos en el contexto de la ecuación no lineal de Schrödinger con potencial lineal. El método AKNS, inicialmente propuesto para determinar la integrabilidad de ecuaciones no lineales, permite establecer que la ecuación no lineal de Schrödinger con potencial lineal es completamente integrable. Posteriormente, las soluciones solitónicas se pueden obtener utilizando la transformada de Bäcklund.

El método AKNS (Ablowitz, Kaup, Newell y Segur, 1973) consiste de unos pasos: (1) establecer un problema de dispersión lineal donde la solución de la ecuación no lineal juega el papel del potencial; (2) elegir la dependencia temporal de las funciones propias de tal manera que los valores propios permanezcan invariantes en el tiempo; (3) resolver el problema de dispersión directa en el tiempo inicial y determinar la dependencia temporal de los datos de dispersión; y (4) resolver el problema de dispersión inversa en tiempos posteriores conociendo los valores propios discretos correspondientes a la dependencia temporal de los otros datos de dispersión reconstruir el potencial. En nuestro trabajo presentamos un esquema AKNS generalizada con los valores propios que tienen una dependencia de tiempo basando en referencias (Serkin y Hasegawa, 2000; Serkin, Hasegawa y Belyaeva, 2007; Serkin, Hasegawa y Belyaeva, 2010; Belyaeva y Serkin, 2011). Demostramos las propiedades de los solitones no autónomos en el marco del modelo generalizado de la NLSE con el potencial lineal.

$$i\frac{\partial Q}{\partial T} + \frac{D(T)}{2}\frac{\partial^2 Q}{\partial x^2} + R(T)|Q|^2Q - 2\lambda(T)xQ = 0 \quad (3.1)$$

Donde $D(T)$, $R(T)$ y $\lambda(T)$ son los coeficientes de la dispersión, la no linealidad y el potencial lineal variables en tiempo.

Formalismo AKNS generalizado

Como primer paso en la construcción del modelo completamente integrable con potencial lineal se introduce el concepto de par de Lax. En el marco

del método de dispersión inversa (Ablowitz y Segur, 1981), la ecuación integrable no lineal surge como condición de compatibilidad del sistema de ecuaciones diferenciales matriciales lineales

$$\begin{aligned}\frac{\partial \psi}{\partial x} &= \hat{F} \psi(x, t), \\ \frac{\partial \psi}{\partial t} &= \hat{G} \psi(x, t)\end{aligned}\tag{3.2}$$

Donde $\psi(x, t) = \{\psi_1, \psi_2\}^T$ es una función compleja de dos componentes que se llama la función de dispersión $\hat{F}$ y $\hat{G}$ son matrices (2x2) con componentes complejos.

Resolviendo la condicion de compatibilidad del sistema (3.2)

$$\frac{\partial \hat{F}}{\partial t} - \frac{\partial \hat{G}}{\partial x} + [\hat{F}, \hat{G}] = 0\tag{3.3}$$

se puede obtener la ecuación de evolución no lineal deseada para el potencial de dispersión $q(x, t)$

Considerando el caso general del método de dispersion inversa con un parámetro espectral dependiente del tiempo $\Lambda(T)$ y las matrices $\hat{F}$ y $\hat{G}$

$$\begin{aligned}\hat{F}(\Lambda; S, T) &= \hat{F} \left\{ \Lambda(T), Q[S(x, t), T]; \frac{\partial Q}{\partial S} \left(\frac{\partial S}{\partial x} \right); \frac{\partial^2 Q}{\partial S^2} \left(\frac{\partial S}{\partial x} \right)^2; \dots; \frac{\partial^2 Q}{\partial S^n} \left(\frac{\partial S}{\partial x} \right)^n \right\}, \\ \hat{G}(\Lambda; S, T) &= \hat{G} \left\{ \Lambda(T), Q[S(x, t), T]; \frac{\partial Q}{\partial S} \left(\frac{\partial S}{\partial x} \right); \frac{\partial^2 Q}{\partial S^2} \left(\frac{\partial S}{\partial x} \right)^2; \dots; \frac{\partial^2 Q}{\partial S^n} \left(\frac{\partial S}{\partial x} \right)^n \right\}\end{aligned}\tag{3.4}$$

dependiente de las coordenadas generalizadas $S = S(x, T)$ y $T(t) = t$, donde la función $Q[S(x, t), T]$ y dependen de los potenciales de dispersión $Q(S, T)$ y $R(S, T)$ y sus derivadas, correspondientemente. La condicion para la compatibilidad del par de ecuaciones diferenciales lineales (3.3) toma la forma

$$\frac{\partial \hat{F}}{\partial T} + \frac{\partial \hat{F}}{\partial S} \frac{\partial S}{\partial T} - \frac{\partial \hat{G}}{\partial S} \frac{\partial S}{\partial x} + [\hat{F}, \hat{G}] = 0\tag{3.5}$$

donde

$$\begin{aligned}\hat{F} &= -i\Lambda(T)\hat{\sigma}_3 + \hat{U}\hat{\phi}, \\ \hat{G} &= \begin{pmatrix} A & B \\ C & -A \end{pmatrix}\end{aligned}\tag{3.6}$$

$\hat{\sigma}_3$ es la matriz de espín de Pauli y las matrices $\hat{U}$ y $\hat{\phi}$ son dadas por:

$$\begin{aligned}\hat{U} &= \sqrt{\sigma}F^\gamma(T) \begin{pmatrix} 0 & Q(S,T) \\ R(S,T) & 0 \end{pmatrix} \\ \hat{\phi} &= \begin{pmatrix} \exp[-i\varphi/2] & 0 \\ 0 & \exp[i\varphi/2] \end{pmatrix}\end{aligned}\tag{3.7}$$

Aquí $F(T)$ y $\varphi(S,T)$ son funciones reales desconocidas que se tienen que encontrar, γ es una constante arbitraria y el parametro $\sigma = \pm 1$ se introduce para separar los solitones brillantes y oscuros.

Los elementos deseados de la matriz $\hat{G}$ (conocidos en la literatura moderna como los elementos AKNS) se pueden construir en la forma $\hat{G} = \sum_{k=0}^3 G_k \Lambda^k$, con el parametro espectral variable en el tiempo $\Lambda(T)$ dado por

$$\Lambda_T = \lambda_0(T) + \lambda_1(T)\Lambda(T)\tag{3.8}$$

donde las funciones dependientes del tiempo $\lambda_0(T)$ y $\lambda_1(T)$ son los coeficientes de expansión de Λ_T en potencias del parámetro espectral $\Lambda(T)$.

Resolviendo el sistema (3.5-3.7), se encuentran los elementos A, B, C de la matriz $\hat{G}$ y dos ecuaciones generales para iQ_T y iR_T .

Al colocar $R = -Q^*$ se genera una reducción, por lo que es fácil encontrar que las ecuaciones de iQ_T y iR_T toman la misma forma si se cumplen las siguientes condiciones:

$$\begin{aligned}a_0 &= -a_0^*, \\ a_1 &= -a_1^*, \\ a_2 &= -a_2^*, \\ a_3 &= -a_3^*, \\ \Lambda_0 &= \lambda_0^*, \\ \lambda_1 &= \lambda_1^*, \\ F &= F^*\end{aligned}\tag{3.9}$$

De acuerdo con las condiciones (3.9), las funciones imaginarias $a_0(T)$, $a_1(T)$, $a_2(T)$ pueden ser definidas como $a_0(T) = i\gamma_0(T)$, $a_1(T) = iV(T)$, $a_2(T) = -iD_2(T)$. El coeficiente de no linealidad está relacionado con el coeficiente de dispersión: $R_2(T) = F^{2\gamma}D_2(T)$, y las funciones $D_2(T)$, $V(T)$, $\gamma_0(T)$ son reales. Los coeficientes $D_2(T)$ y $R_2(T)$ son representados por funciones definidas positivas (para $\sigma = -1$), γ se asume como un número semi entero. Esto permite generar la forma general de la ecuación NLSE con términos de ganancia o absorción Γ , con potencial lineal λ_0 y parabolico Ω

$$i\frac{\partial Q}{\partial T} + \frac{1}{2}D_2(T)\frac{\partial^2 Q}{\partial X^2} + \sigma R_2(T)|Q^2|Q - 2\lambda_0(T)XQ - \frac{1}{2}\Omega^2(T)X^2Q = i\Gamma Q \quad (3.10)$$

Finalmente, se puede escribir la forma general de la ecuación NLSE con potencial lineal y coeficientes de dispersión y no linealidad dependientes del tiempo, si no hay termino de ganancia o absorción ($\Gamma = 0$) y considerando el potencial parabolico como cero ($\Omega = 0$) .

$$i\frac{\partial Q}{\partial T} + \frac{1}{2}D_2(T)\frac{\partial^2 Q}{\partial X^2} + \sigma R_2(T)|Q^2|Q - 2\lambda_0(T)XQ = 0 \quad (3.11)$$

Por lo que gracias al método AKNS significa que la ecuación es completamente integrable y permite soluciones tipo solitón.

Soluciones solitonicas mediante la transformada de Backlund

La Transformada de Bäcklund es una herramienta util en la teoría de ecuaciones no lineales, que permite construir nuevas soluciones a partir de soluciones conocidas. En el contexto de la ecuación no lineal de Schrödinger (NLSE) con potencial lineal, este método proporciona una manera efectiva de generar soluciones solitonicas múltiples, conocidas como soluciones N-solitonicas.

Definición de la transformada de Backlund

Sea $\psi(x, t)$ una solución conocida de la NLSE bajo el potencial lineal dado por la ecuación (3.11). La transformada de Backlund permite generar una nueva solución $\psi'(x, t)$ a partir de $\psi(x, t)$ con la forma

$$\psi'(x, t) = \psi(x, t) + \Delta\psi(x, t) \quad (3.12)$$

donde $\Delta\psi(x, t)$ es una corrección que depende de los parámetros del sistema, tales como el potencial lineal, los coeficientes de dispersión, no linealidad y las propiedades de la solución inicial $\psi(x, t)$. Este término de corrección está ligado al pseudopotencial del sistema, el cual desempeña un papel crucial en la evolución de la solución solitónica.

Ecuaciones de la transformada de Backlund

La forma específica de la transformada para la NLSE se obtiene resolviendo las ecuaciones de Backlund, que establecen la relación entre la solución original $\psi(x, t)$ y la nueva solución $\psi'(x, t)$. Estas ecuaciones se pueden escribir de la siguiente manera:

$$\begin{aligned} \frac{\partial\psi'}{\partial x} &= \frac{\partial\psi}{\partial x} + F(\psi, \psi', \lambda), \\ \frac{\partial\psi'}{\partial t} &= \frac{\partial\psi}{\partial t} + G(\psi, \psi', \lambda) \end{aligned} \quad (3.13)$$

donde F y G son funciones que dependen de la solución $\psi(x, t)$ y el parámetro espectral λ , que caracteriza la forma de la corrección solitónica.

La elección del parámetro λ está relacionada directamente con las propiedades de la nueva solución. Por ejemplo en el caso de solitones brillantes, λ puede estar asociado con la amplitud y la velocidad del soliton generado.

Construcción de soluciones solitónicas

Una vez que se han resuelto las ecuaciones de Backlund, es posible construir explícitamente las soluciones solitónicas. Para una solución de un solo solitón, la transformada de Backlund genera una corrección de la forma:

$$\Delta\psi(x, t) = 2\frac{\partial}{\partial x}\ln\left(1 + \frac{e^{\theta(x, t)}}{\psi(x, t)}\right) \quad (3.14)$$

donde $\theta(x, t)$ es una función auxiliar que depende del tiempo y el espacio, y que determina el perfil del soliton añadido. Para un soliton brillante, esta función podría tener la forma:

$$\theta(x, t) = \eta_0 x - \kappa_0 t \quad (3.15)$$

donde η_0 y κ_0 son parámetros relacionados con la amplitud y velocidad del solitón.

Soluciones N-solitonicas

El método de Bäcklund no se limita a generar una solución solitonica individual. Al aplicar de manera iterativa la transformada, es posible obtener soluciones de múltiples solitones. Para generar una solución de dos solitones, se aplica la transformada sobre la solución de un solitón:

$$\psi_{2soliton}(x, t) = \psi_{1soliton} + \Delta\psi_1(x, t) \quad (3.16)$$

Cada iteración añade un nuevo solitón al sistema, respetando la integrabilidad de la ecuación y preservando las propiedades esenciales de la solución original. En particular, las soluciones múltiples muestran interacción elástica, lo que significa que después de colisionar, los solitones conservan su forma y velocidad.

Entonces las soluciones solitónicas de la Ec.(3.11) se puede obtener aplicando la transformación de Bäcklund (Chen, 1974) y la relación de recurrencia (Rangwala y Rao, 1985; Serkin, Hasegawa y Belyaeva 2010).

$$Q_n(x, T) = -Q_{n-1}(x, T) - \frac{4\eta_n \tilde{\Gamma}_{n-1}(x, T)}{1 + |\tilde{\Gamma}_{n-1}(x, T)|^2} \sqrt{\frac{D(T)}{R(T)}} \exp[-i\Theta x^2/2] \quad (3.17)$$

Estas soluciones tienen una fase adicional autoinducida (un "chirp")

$$\varphi(x, T) = \Theta(T)x^2/2 + \varphi : 0(T = 0) \quad (3.18)$$

donde el perfil autoinducido $\Theta(T)$ se genera en concordancia con los coeficientes variables del sistema:

$$\Theta(T) = -\frac{W[R(T), D(T)]}{D^2(T)R(T)} \quad (3.19)$$

y depende del Wronskiano $W(R, D) = RD'_T - DR'_T$ para la ecuación (3.11) los coeficientes están relacionados como:

$$\frac{D(T)}{D_0} = \frac{R(T)}{R_0} \exp\left[-\frac{\Theta_0 D_0}{R_0} \int_0^T R(\tau) d\tau\right] \quad (3.20)$$

Ecuación (3.17) conecta las soluciones para (n-1) y n solitones mediante el llamado pseudopotencial $\tilde{\Gamma}(x, T) = \frac{\psi_1(x, T)}{\psi_2(x, T)}$ para las funciones $\psi(x, T) = (\psi_1 \psi_2)^T$ de dispersión del solitón (n-1).

Propiedades de las soluciones generadas

Las soluciones obtenidas mediante la transformada de Backlund presentan una serie de características interesantes. Primero, los solitones no autónomos muestran una autoinducción de fase o "chirp", que es una modulación en su frecuencia debido a la dependencia temporal de los coeficientes del sistema. Esta modulación afecta tanto a la amplitud como a la forma del solitón:

$$\phi(x, T) = \phi_0 e^{i\Theta(x, T)} \quad (3.21)$$

donde $\Theta(x, T)$ representa la fase autoinducida.

Además, los solitones muestran comportamientos adaptativos al potencial lineal y a las variaciones de dispersión y no linealidad, ajustando su velocidad y amplitud en función del medio externo.

Condiciones de frontera y condiciones iniciales

De la relación (3.17) se obtienen el 1 y 2 soliton de los cuales son la base de estudio en las condiciones iniciales y de frontera consideradas para el aprendizaje en la red neuronal.

Donde se consideraron los parametros

$$\begin{aligned}
R(t) &= R_0 = 1 \\
D(t) &= D_0 \exp(-c_0 t), \text{ donde } D_0 = 1 \\
\Theta(t) &= \Theta_0 \exp(c_0 t) \text{ donde } \Theta_0 = 1. \\
\text{y el coeficiente del potencial } \lambda_0(t) &= \text{Const.}
\end{aligned} \tag{3.22}$$

El parametro c_0 debe ser un numero constante el cual nunca puede ser igual a cero ya que la solucion soliton estaria indefinida por una divición con 0 lo cual permite obtener las soluciones 1 y 2 soliton considerando $n = 1$ y $n = 2$

Para $n = 1$

$$\begin{aligned}
Q_1(x, t) &= 2\eta_{01} \sqrt{D_0 \exp(c_0 t)} \operatorname{sech}[\xi_1(x, t)] \times \\
&\exp\left[-\frac{i}{2} \Theta_0 \exp(c_0 t) x^2 - i\chi_1(x, t)\right]
\end{aligned} \tag{3.23}$$

Para $n = 2$

$$\begin{aligned}
Q_2(x, t) &= 4\sqrt{D_0 \exp(-c_0 t)} \frac{\mathbb{N}(x, t)}{\mathbb{D}(x, t)} \times \\
&\exp\left[-\frac{i}{2} \Theta_0 \exp(c_0 t) x^2\right]
\end{aligned} \tag{3.24}$$

donde $\mathbb{N}(x, t)$ y $\mathbb{D}(x, t)$ estan dadas por

$$\begin{aligned}
\mathbb{N} &= \cosh(\xi_2) \exp(-i\chi_1) \times [(\kappa_2 - \kappa_1)^2 + \\
&2i\eta_2(\kappa_2 - \kappa_1) \tanh(\xi_2) + \eta_1^2 - \eta_2^2] + \\
&\eta_2 \cosh(\xi_1) \exp(-i\chi_2) \times [(\kappa_2 - \kappa_1)^2 - \\
&2i\eta_1(\kappa_2 - \kappa_1) \tanh(\xi_1) - \eta_1^2 + \eta_2^2], \\
\mathbb{D} &= \cosh(\xi_1 + \xi_2) \left[(\kappa_2 - \kappa_1)^2 + \right. \\
&(\eta_2 - \eta_1)^2 \left. \right] + \cosh(\xi_1 - \xi_2) \left[(\kappa_2 - \right. \\
&\left. \kappa_1)^2 + (\eta_2 + \eta_1)^2 \right] - 4\eta_1 \eta_2 \cos(\chi_2 - \chi_1)
\end{aligned} \tag{3.25}$$

con

$$\xi_i(x, t) = 2\eta_{0i}x\exp(C_0t) + 4D_0\eta_{0i} \left\{ \frac{\kappa_{0i}}{C_0} [\exp(C_0t) - 1] + \frac{\lambda_0}{C_0} \left[\frac{\exp(C_0t) - 1}{C_0} - t \right] \right\} \quad (3.26)$$

$$\begin{aligned} \chi_i(x, t) = & 2\kappa_{0i}x\exp(C_0t) + 2D_0(\kappa_{0i}^2 - \eta_{0i}^2) \frac{\exp(2C_0t) - 1}{2C_0} + 2x \frac{\lambda_0}{C_0} [\exp(C_0t) - 1] + \\ & 4D_0\kappa_{0i} \frac{\lambda_0}{C_0} \left[\frac{\exp(C_0t) - 1}{C_0} - t \right] + 2D_0 \left(\frac{\lambda_0}{C_0} \right)^2 \left[\frac{\exp(C_0t) - \exp(-C_0t)}{C_0} - 2t \right] \end{aligned} \quad (3.27)$$

las amplitudes

$$\eta_i(t) = \frac{D_0 R(t)}{R_0 D(t)} \eta_{0i} \quad (3.28)$$

y las velocidades

$$\kappa_i(t) = \frac{D_0 R(t)}{R_0 D(t)} \left[\kappa_{0i} + \frac{R_0}{D_0} \int_0^t \frac{\lambda_0(\tau) D(\tau)}{R(\tau)} d\tau \right] \quad (3.29)$$

siendo los argumentos y fases $\xi_i(x, t)$, $\chi_i(x, t)$ que estan relacionadas con las amplitudes $\eta_i(t)$ y las velocidades $\kappa_i(t)$, donde κ_{0i} y η_{0i} corresponden a la velocidad inicial y la amplitud del i-esimo soliton.

Se observan en las figuras 3.1 y 3.2 la dinamica de estos solitones

3.2. Implementación del Método PINN

El objetivo de PINN es aproximar la solución de un sistema de una o más ecuaciones diferenciales, codificando explícitamente la formulación de la ecuación diferencial en la red neuronal. Sin perdida de la generalidad, PINN resuelve la ecuación no lineal:

$$u_t(x) = Nu(x) = 0, \quad x \in \Omega, \quad t \in [0, T] \quad (3.30)$$

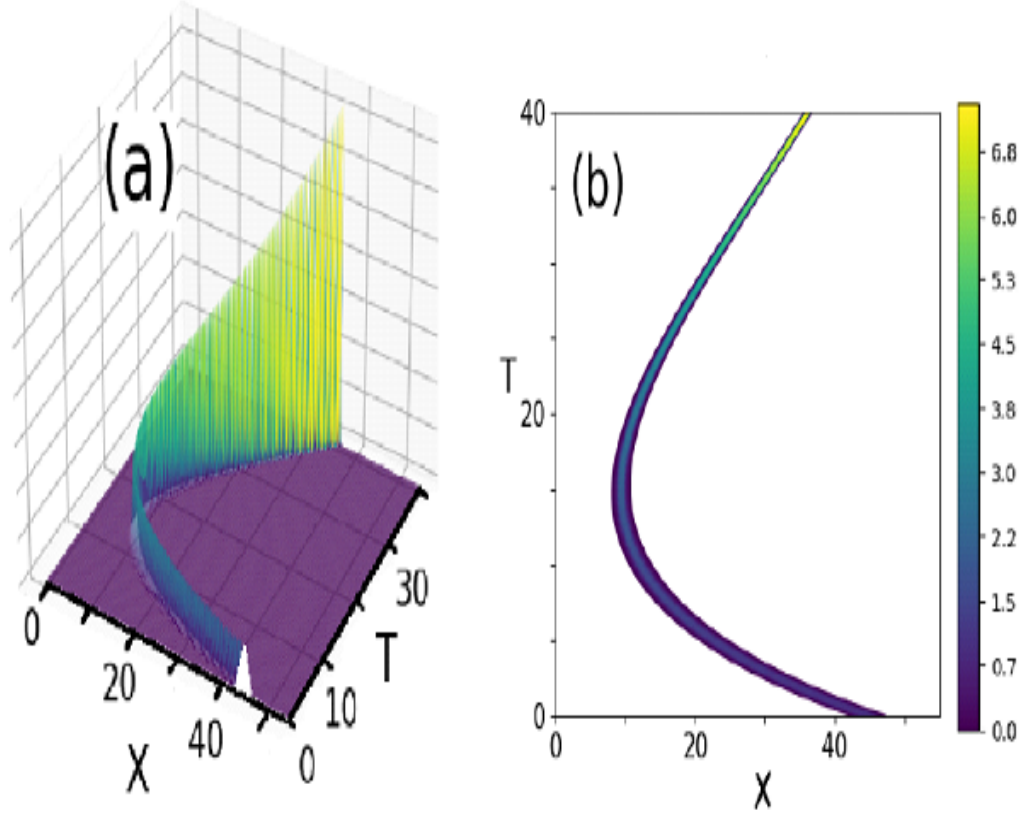


Figura 3.1: Evolución del solitón brillante no autónomo representado por la Ecuación (3.23) con parámetros del modelo $c_0 = 0.05$, $\lambda_0 = -0.2$, $\eta_{01} = 0.5$, $\kappa_{01} = 3$. Donde (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

Donde u es la solución del sistema, u_t es la derivada con respecto al tiempo t en el periodo $[0, T]$, N es un operador diferencial no lineal, x es una posible variable multidimensional, definida sobre el dominio Ω .

En su formulación básica, los PINN combinan dos redes: una red aproximada o sustituta y una red residual. La red aproximador/sustituto se capacita y luego proporciona una solución $\tilde{u}$ en un punto de entrada dado (x, y) , llamado punto de colocación, en el dominio de simulación. La red residual codifica las ecuaciones gobernantes y es la característica distintiva de los

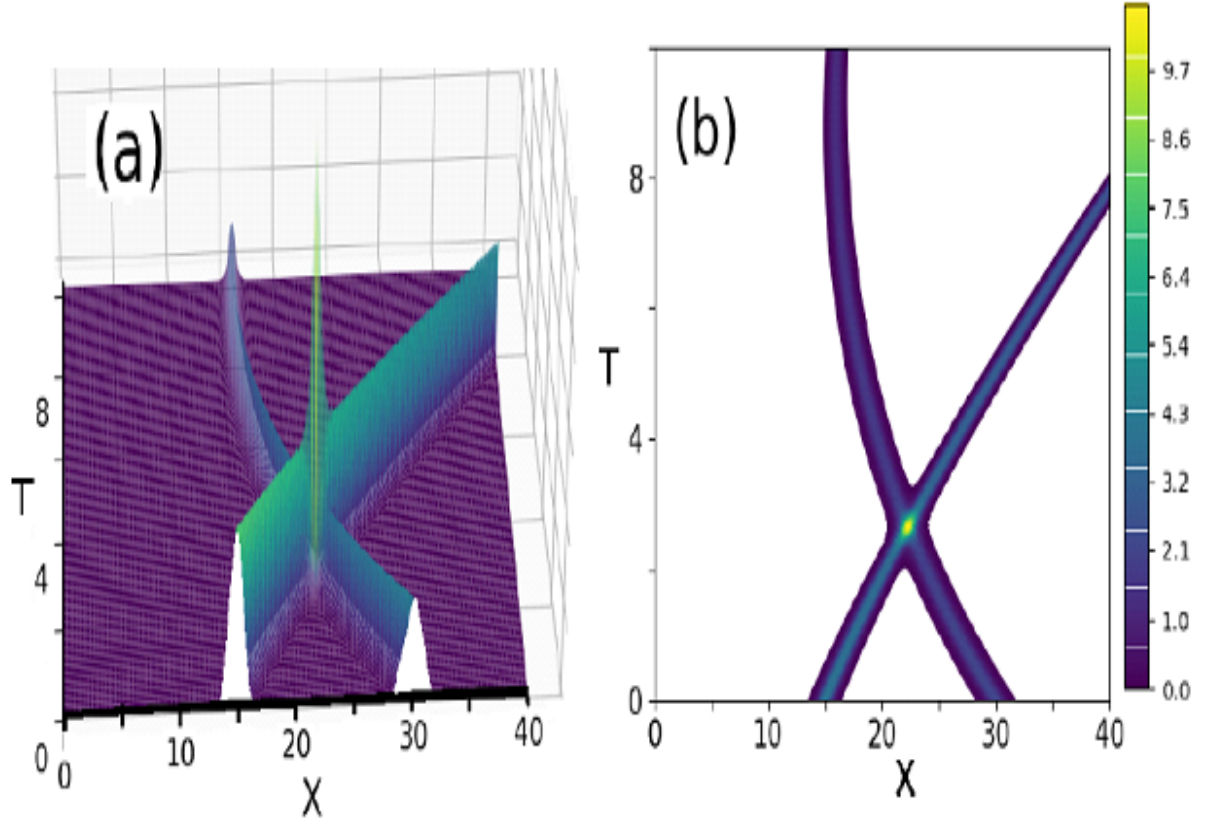


Figura 3.2: Evolución de dos solitones brillantes no autónomos representados por la Ecuación (3.24) con parámetros del modelo: $c_0 = 0.05$, $\lambda_0 = -0.2$, $\eta_{01} = 1$, $\eta_{02} = 0.75$, $\kappa_{01} = -1.2$, $\kappa_{02} = 1.8$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

PINN. La red residual no está entrenada y su única función es proporcionar a la red aproximada/sustituta el residual (función de pérdida en terminología de aprendizaje profundo (DL)).

La red aproximada/sustituta es una red neuronal de avance: procesa una entrada x a través de l capas de unidades (llamadas neuronas). La red aproximada/sustituta expresa aplicaciones lineales afines (Z) entre unidades y funciones de activación escalares no lineales (a) dentro de las unidades:

$$\tilde{u}(x) = Z_l \circ a \circ Z_{l-1} \circ a \dots a \circ Z_2 \circ a \circ Z_1(x) \quad (3.31)$$

En DL las funciones de activacion más utilizadas son las funciones de unidad lineal rectificada (ReLU), tanh, swish, seno y sigmoidea.

Los mapas afines Z se caracterizan por los pesos y sesgos de la red aproximada/sustituta:

$$Z_l x_l = W_l x_l + b_l \quad (3.32)$$

Donde W_l es una matriz de pesos para las capas l y b es el vector de sesgo. En PINNs, los valores de peso se inicializan utilizando el procedimiento Xavier. En PINN, se minimiza el error cuadrático medio (MSE) del residual

$$MSE_r = \frac{1}{N_{x_i, y_i}} \sum |r(x_i, y_i)|^2 \quad (3.33)$$

Donde N_{x_i, y_i} es el número de puntos de colocación y la función r es la función de pérdida que consta de la solución aproximada a la que se le restan los valores tomados por los datos exactos que entrenan a la red. En la fase de entrenamiento, un proceso de optimización dirigido a la minimización residual determina los pesos y sesgos de la red sustituta. Normalmente se usan dos optimizadores seguidos: primero el optimizador Adam y luego un optimizador Broyden-Fletcher-Goldfarb-Shanno (BFGS). BFGS utiliza la matriz de Hesse (curvatura en un espacio altamente dimensional) para calcular la dirección de optimización y proporciona resultados más precisos. Sin embargo si se usa sin el optimizador Adam, puede converger rápidamente a mínimos locales sin salir de ellos. Todos los marcos PINN principales están escritos en Python y se vasan en TensorFlow o PyTorch para expresar la arquitectura de la red neuronal y explotar la diferenciación automática utilizada en la red residual.

Detalles de la arquitectura de la red neuronal utilizada

Para la ecuación no lineal de schrodinger con potencial lineal la formulación en la PINN es representada por su parte real e imaginaria de la siguiente manera:

$$\begin{aligned} f_v &= -v_t + D(t)u_{xx} + R(t)(u^2 + v^2)u - 2(\lambda_0(t))xu \\ f_u &= u_t + D(t)v_{xx} + R(t)(u^2 + v^2)v - 2(\lambda_0(t))xv \end{aligned} \quad (3.34)$$

donde las funciones u y v representan la parte real e imaginaria de la ecuación.

Para las condiciones de frontera e iniciales se considera un tiempo pequeño en la dinamica del soliton para que la red neuronal no tenga demasiados problemas con la complejidad de la solución y poder maximisar el aprendizaje por lo que se toma $x \in [-a, a]$ y $t \in [0, 1]$.

La función de activación utilizada es la tanh ya que es la que se usa mas comunmente para este tipo de problemas no lineales; usando ademas los optimizadores Adam y el LBFGS con un número considerable de iteraciones de ambos dependiendo el analisis y con las consideraciones aplicadas y estudiadas en los parametros de la solucion, que se detallan mas adelante en el capitulo de los resultados.

La arquitectura del programa PINN fue realizada en Python utilizando su estructura TensorFlow se uso el metodo LHS (Latin Hypercubic sampling) para que la red tomara los valores aleatorios de los proporcionados de la data los cuales consideramos usar los valores N_0 , N_b y N_f que son respectivamente el número de puntos de datos considerados aleatoriamente de las condiciones iniciales y de frontera de la solución, N_f son los datos aleatorios que tomara la red para el entrenamiento generados por el metodo de sampleo LHS, utilizando un cpu i5-12400, se probaron configuraciones varias de capas y numero de neuronas de las cuales en el capitulo de resultados se dan mas detalles.

La estructura del programa usado se muestra en el apendice A

Capítulo 4

Resultados y Discusión

4.1. Soluciones

Para el 1 soliton empezamos analizando la solución (3.23) con las condiciones de frontera y temporales $x \in [-5, 5]$ $t \in [0, 1]$ con la siguiente grafica que describe su dinamica

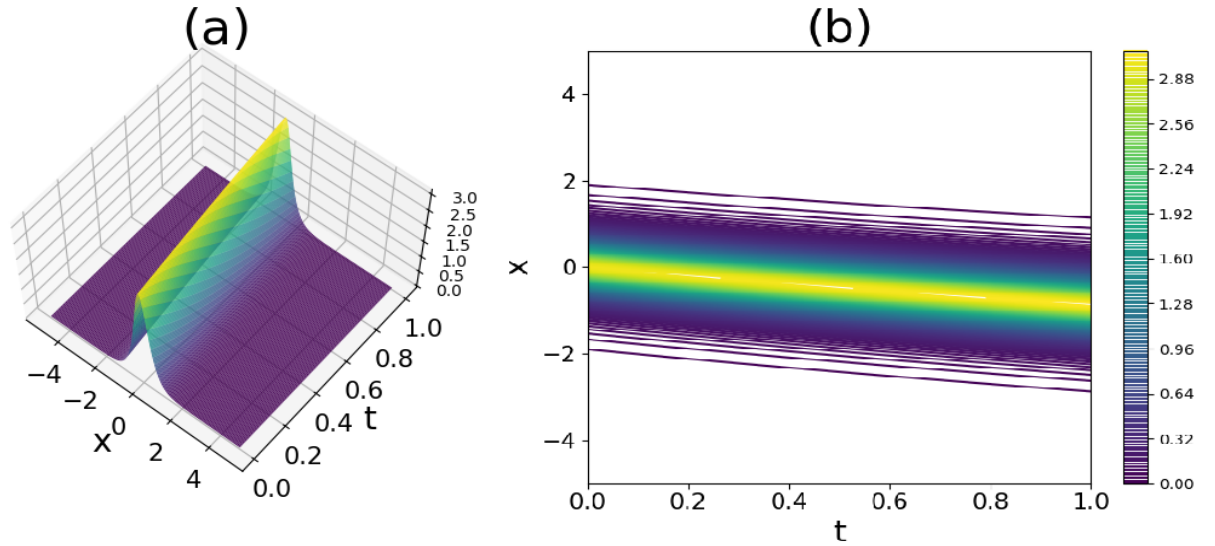


Figura 4.1: Dinamica de 1 soliton con los parametros $c_0 = 0.05$, $\lambda_0 = -0.2$. $\eta_{01} = 1.5$, $\kappa_{01} = 0.5$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

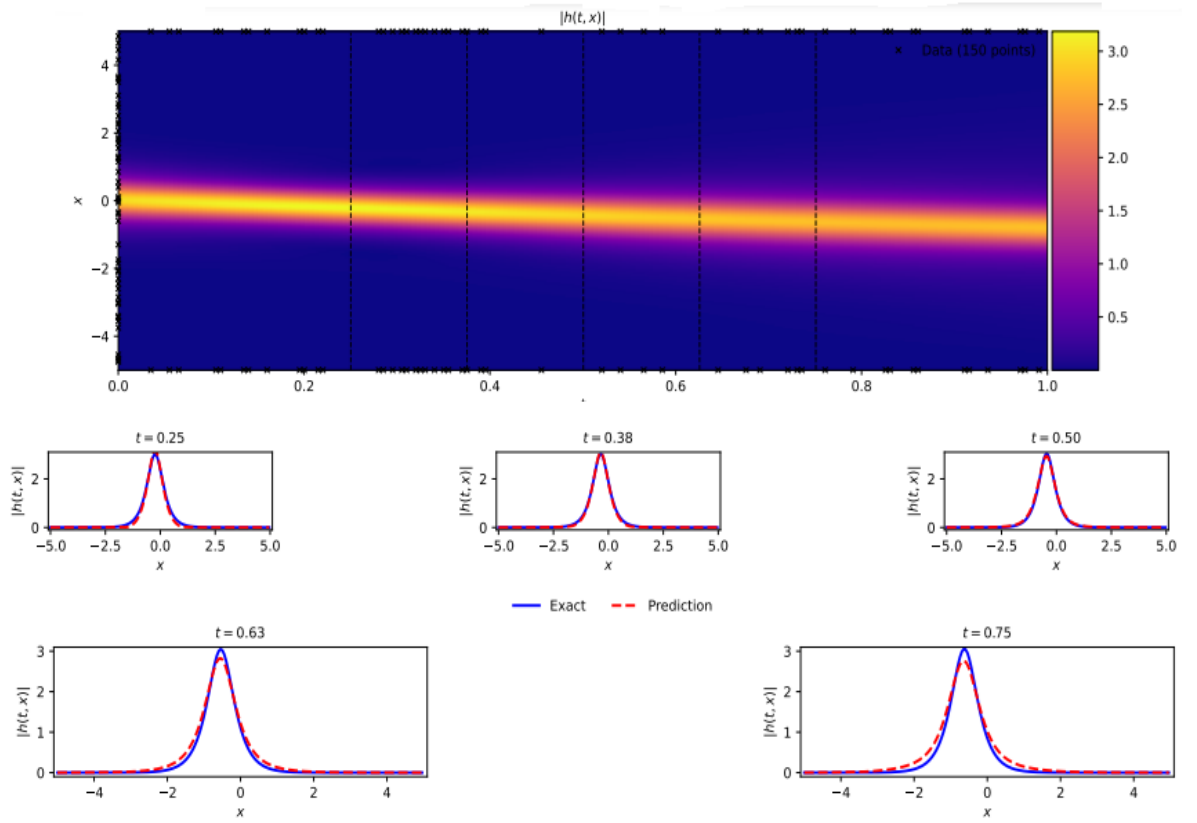


Figura 4.2: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 12291s (aprox. 3.4hrs)

Esta solución es el 1 soliton con velocidad, bajo la influencia del potencial y cuenta con una fase autoinducida o “chirp” la red neuronal obtuvo los siguientes resultados mostrados a continuación figuras 4.2, 4.3 y 4.4.

La configuración de las capas y neuronas fue la siguiente: 4 capas y 100 neuronas por capa ([2, 100, 100, 100, 100, 2]) donde se observa que la red obtuvo resultados de aprendizaje pero no son lo suficientemente buenos para considerarlos unos resultados aceptables se observa que en la figura 4.4 el optimizador Adam a pesar de realizar 10000 iteraciones no logra reducir el error en a un valor menor de 1×10^{-2} por lo que se realizo un analisis de la solución para detectar cual es el parametro que mas le causa problemas a la red.

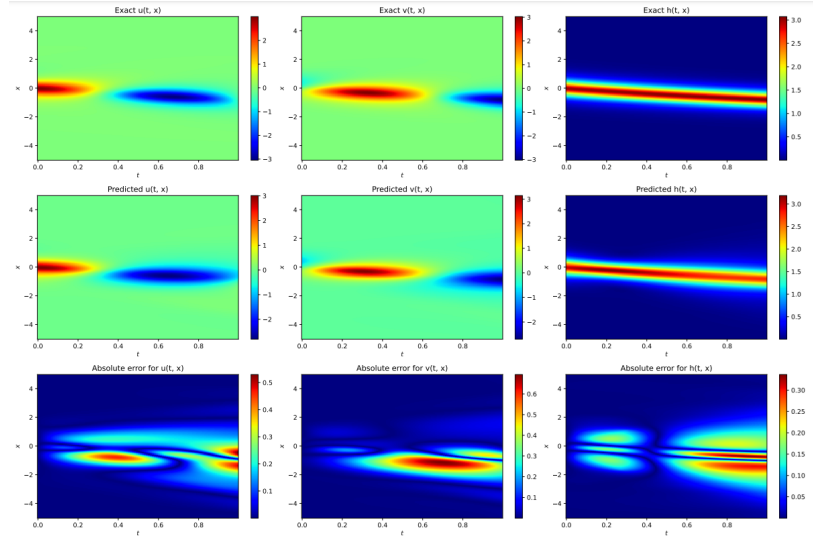


Figura 4.3: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u : $1.913404e-01$, Error v : $2.663007e-01$, Error h : $1.045360e-01$

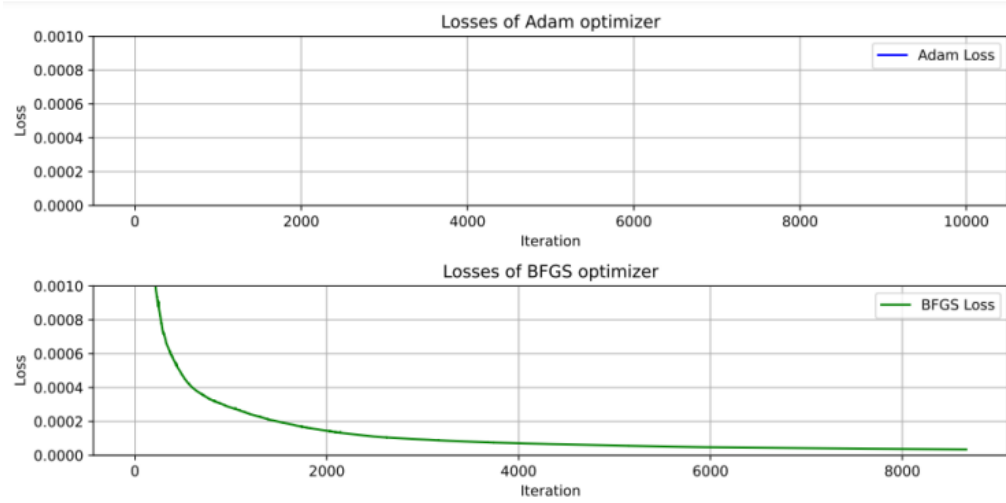


Figura 4.4: grafica del error de los optimizadores por iteración donde los valores minimos obtenidos fueron: Iteration 9900, Loss Adam: 0.003535050665959716 y BFGS-B: $3.453735e-05$

Consideramos los parametros principales que pueden estar causando problemas al aprendizaje de la red como los siguientes: que el soliton tenga una velocidad, que el soliton tiene una dispersion no constante, que esta bajo la influencia del potencial lineal y que tiene una funcion de fase autoinducida (“chirp”); estos parametros fueron modificados en la solución para mejorar el aprendizaje de la red y analizar su influencia en el aprendizaje los resultados se muestran a continuación, en las características como la dispersión o el chirp que tienen este parametro se consideran simplificados o eliminados de la solución con el proposito analitico en sus efectos en el aprendizaje.

Solucion mas simple posible

Esta solución es el soliton en su version mas simple ya que no tiene velocidad, su dispersión es constante ($D(t) = 1$), no hay fase autoinducida y no sufre influencia del potencial lineal

$$q(x, t) = 2\eta_{01} \text{sech}[\xi_1(x, t)] \exp \left[-i\chi_1(x, t) \right] \quad (4.1)$$

su grafica es la figura 4.5.

Los resultados obtenidos por la PINN se muestran en las figuras 4.6, 4.7 y 4.8.

Los resultados de la solución mas simple nos demuestran que al simplificar los parametros de la solución el aprendizaje mejora mucho por lo que las siguientes graficas analizamos los parametros y su influencia en el aprendizaje.

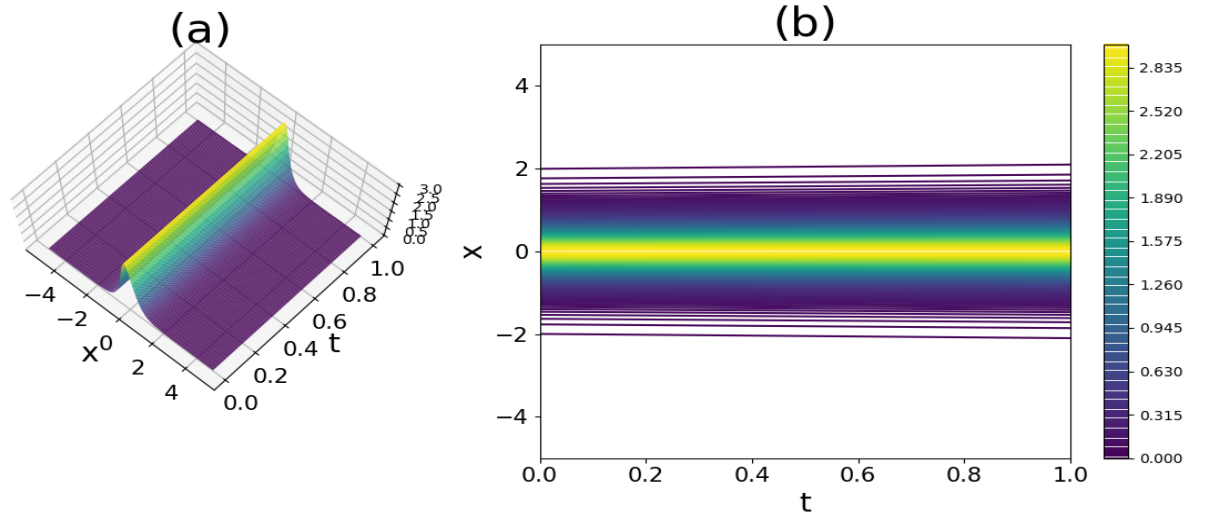


Figura 4.5: Dinamica de 1 soliton con los parametros $c_0=0$, $\lambda_0=0$, $\eta_{01}=1.5$, $\kappa_{01}=0$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

Solución con solo velocidad y parametro $c_0 = 0,05$

Esta solución es la misma que (4.1) con la diferencia de que el soliton tiene una velocidad en su movimiento su grafica es la figura 4.9.

La PINN obtuvo lo siguiente en las figuras 4.10, 4.11 y 4.12

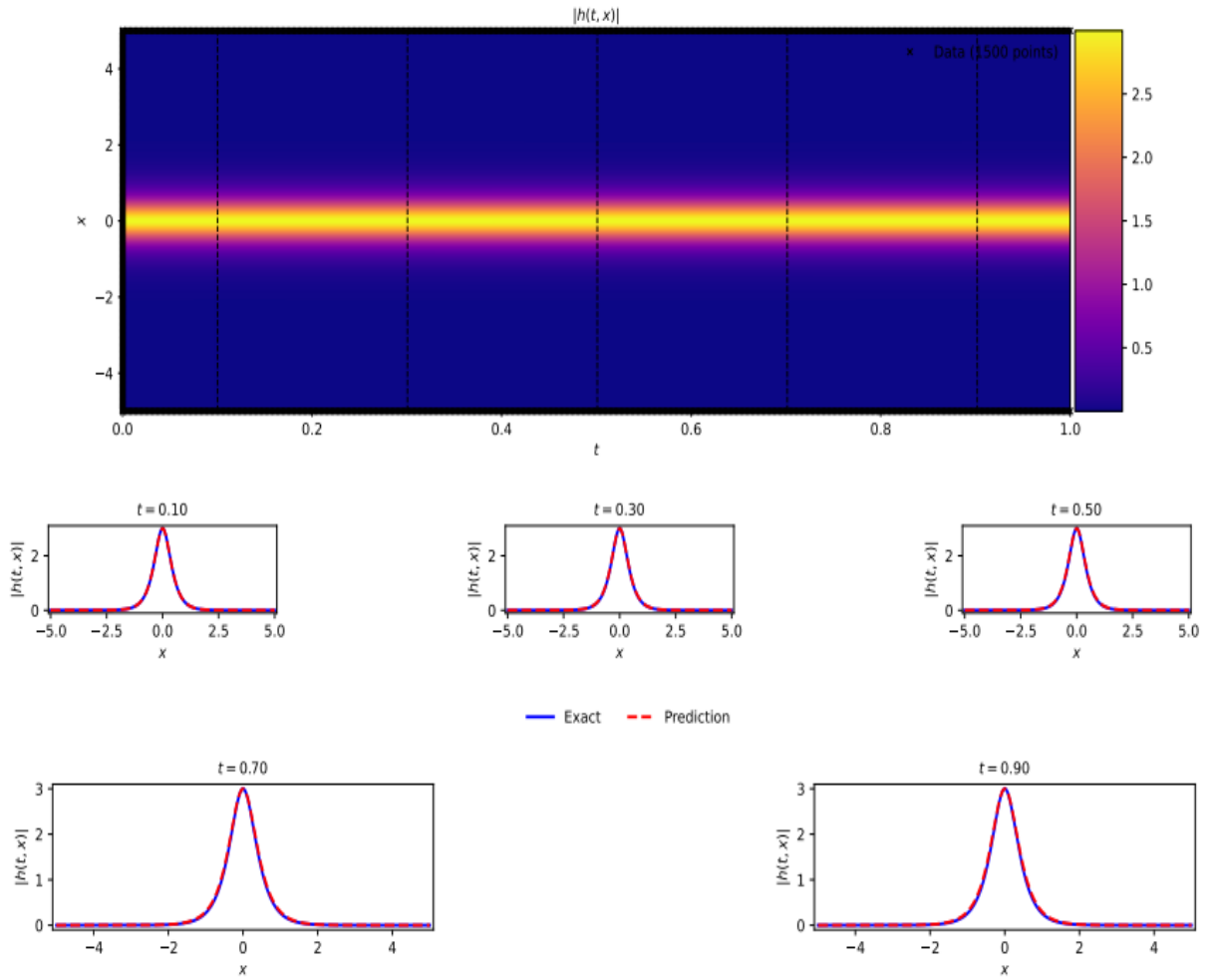


Figura 4.6: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 21825s (aprox. 6hrs)

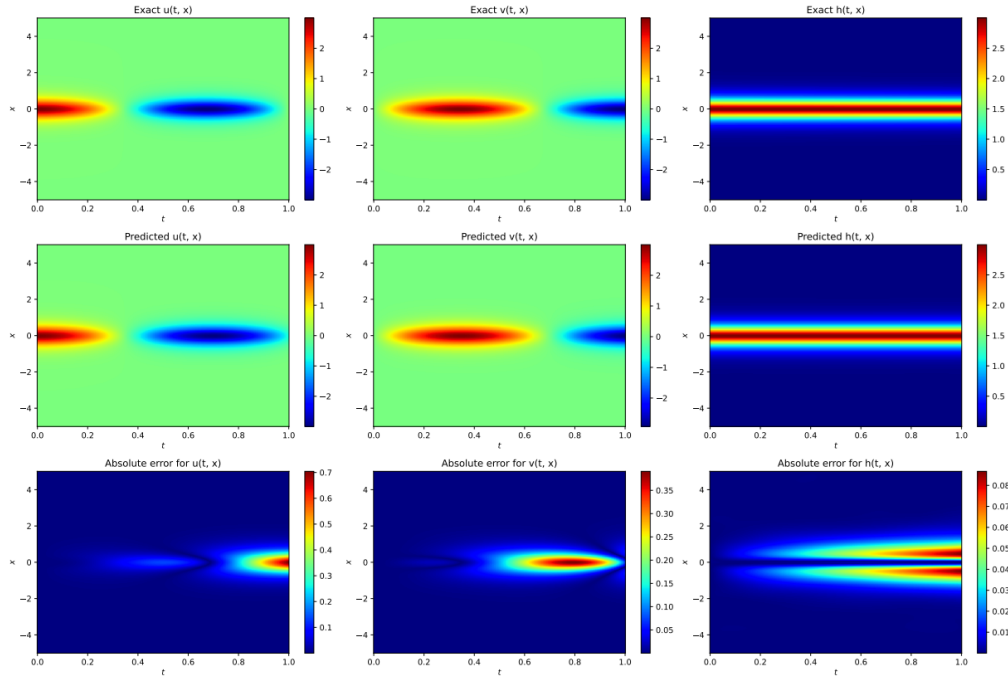


Figura 4.7: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u: 1.192680e-01, Error v: 9.580201e-02, Error h: 2.270304e-02

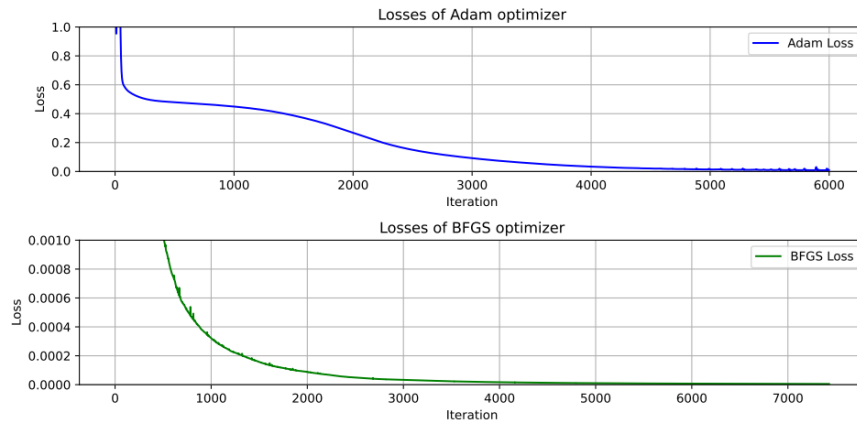


Figura 4.8: grafica del error de los optimizadores por iteración

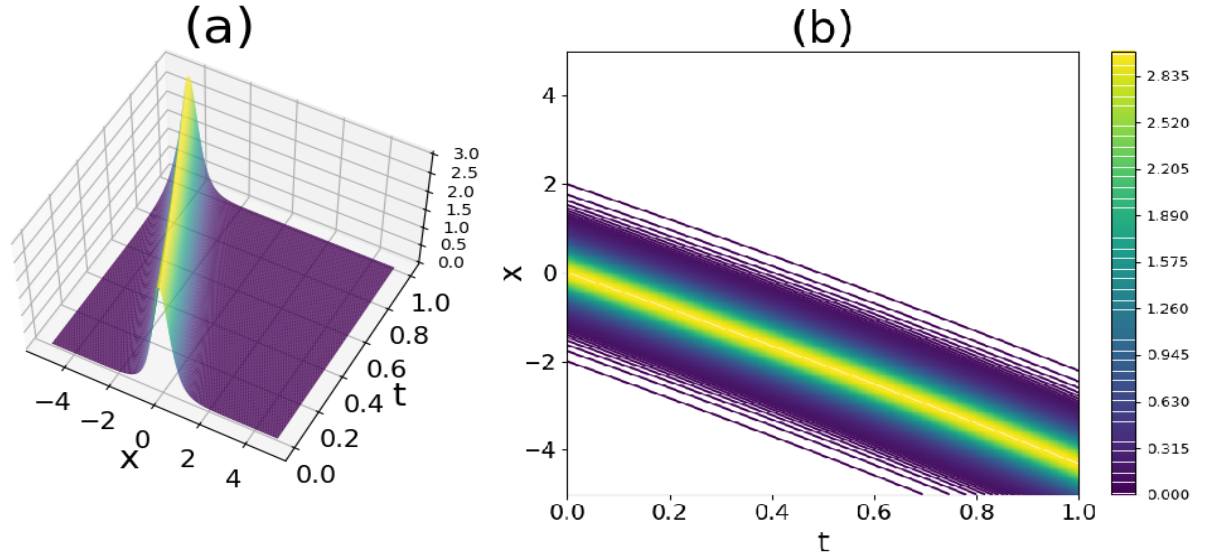


Figura 4.9: Dinamica de 1 soliton con los parametros $c_0 = 0.05$, $\lambda_0 = 0$. $\eta_{01} = 1.5$, $\kappa_{01} = 2$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

Solución con solo potencial y parametro $c_0 = 0,05$

Esta solución es la misma que (4.1) con la diferencia de que el soliton solo esta bajo la influencia del potencial lineal su grafica es la figura 4.13

La PINN obtuvo lo siguiente en las figuras 4.14, 4.15 y 4.16

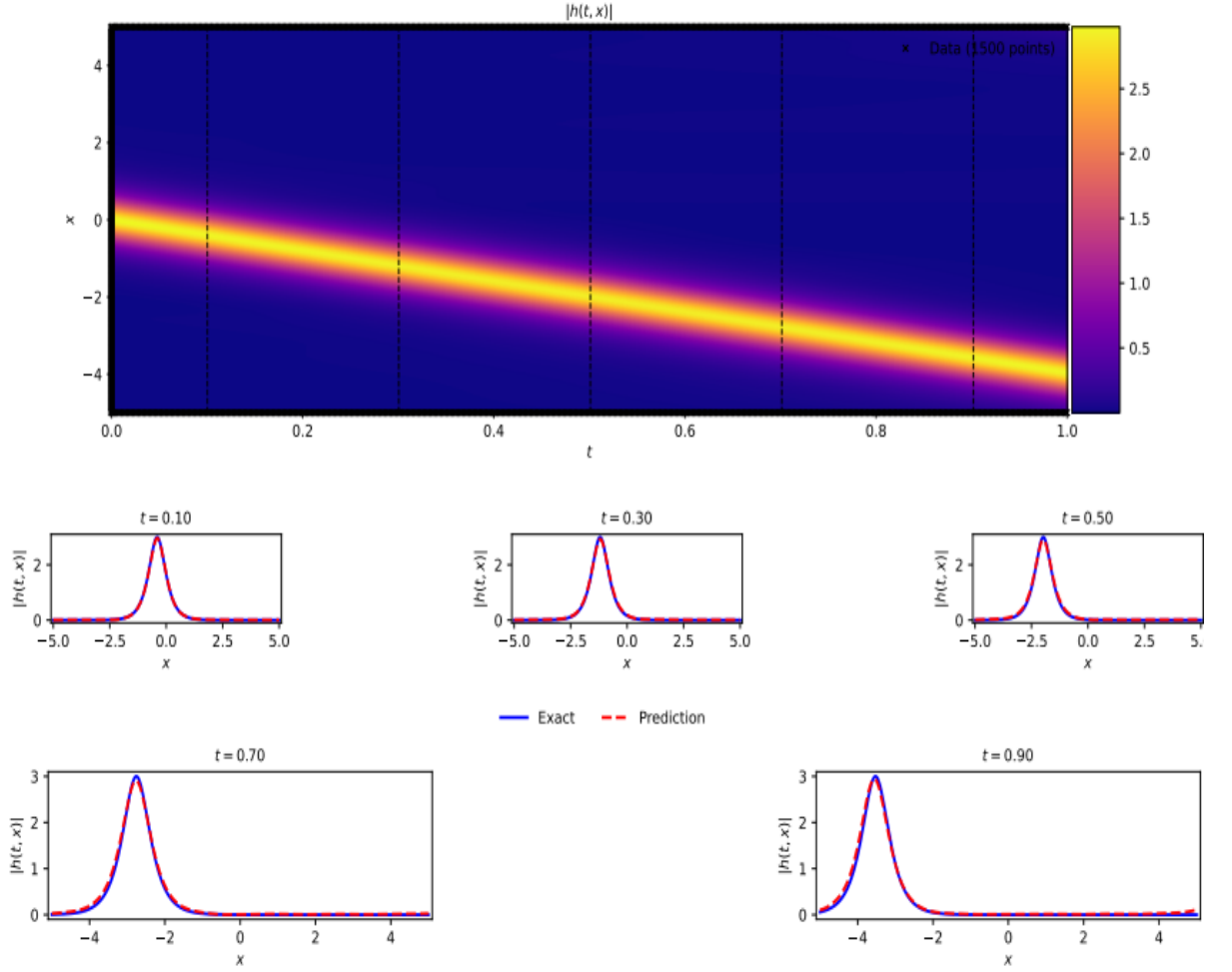


Figura 4.10: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 25155s (aprox. 6.9hrs)

Solución con solo dispersión y parametro $c_0 = 0,05$

Esta solución el soliton solo tiene una dispersión de la forma $D_0 \exp(c_0 t)$ influyendo su dinamica

$$q(x, t) = 2\eta_{01} \sqrt{D_0 \exp(c_0 t)} \operatorname{sech}[\xi_1(x, t)] \exp\left[-i\chi_1(x, t)\right] \quad (4.2)$$

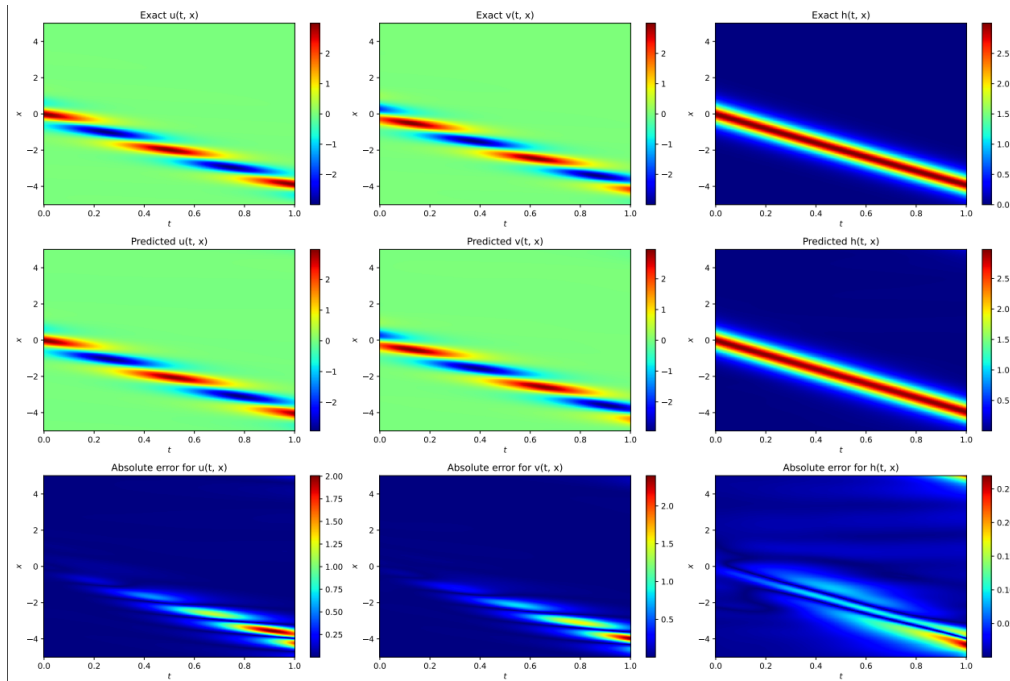


Figura 4.11: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u: 3.952913e-01, Error v: 3.958652e-01, Error h: 5.009197e-02

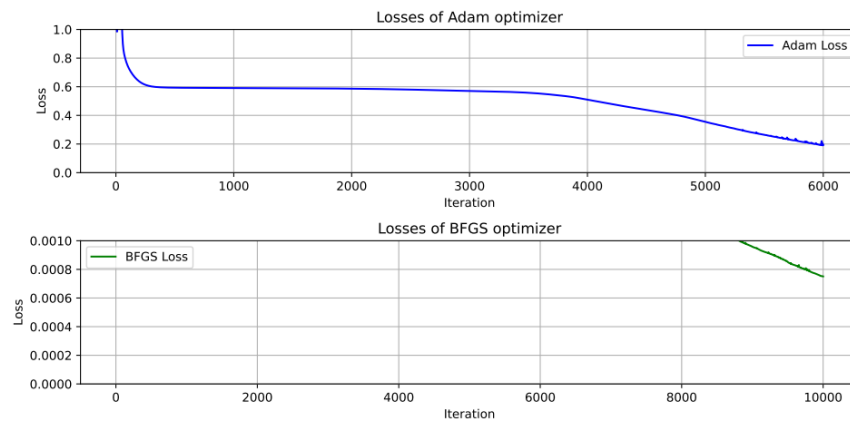


Figura 4.12: grafica del error de los optimizadores por iteración

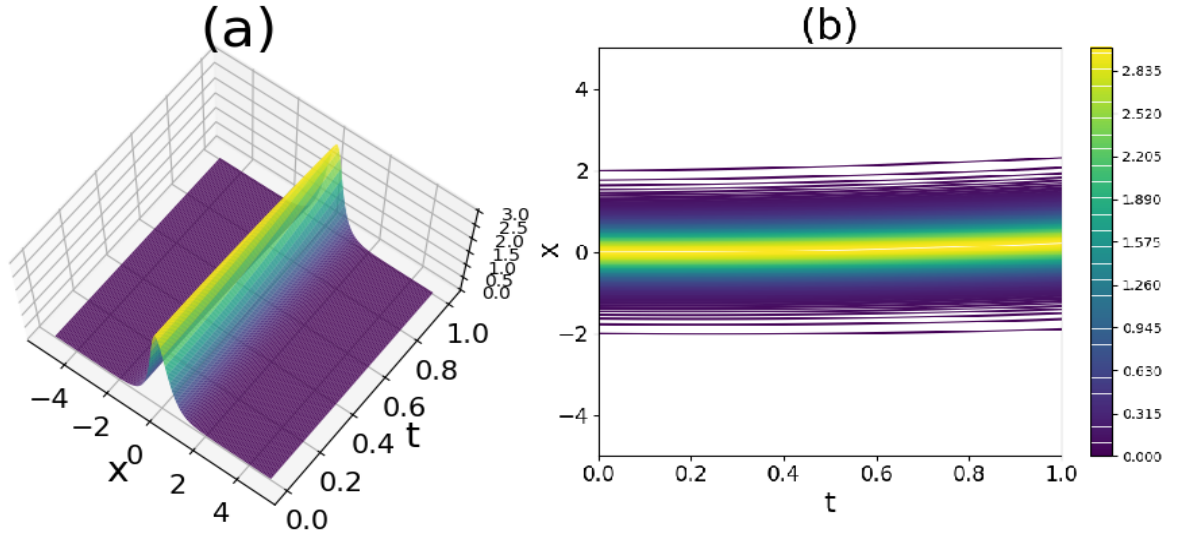


Figura 4.13: Dinamica de 1 soliton con los parametros $c_0 = 0.05$, $\lambda_0 = -.2$. $\eta_{01} = 1.5$, $\kappa_{01} = 0$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

su grafica es la figura 4.17

La PINN obtuvo lo siguiente en las figuras 4.18, 4.19 y 4.20

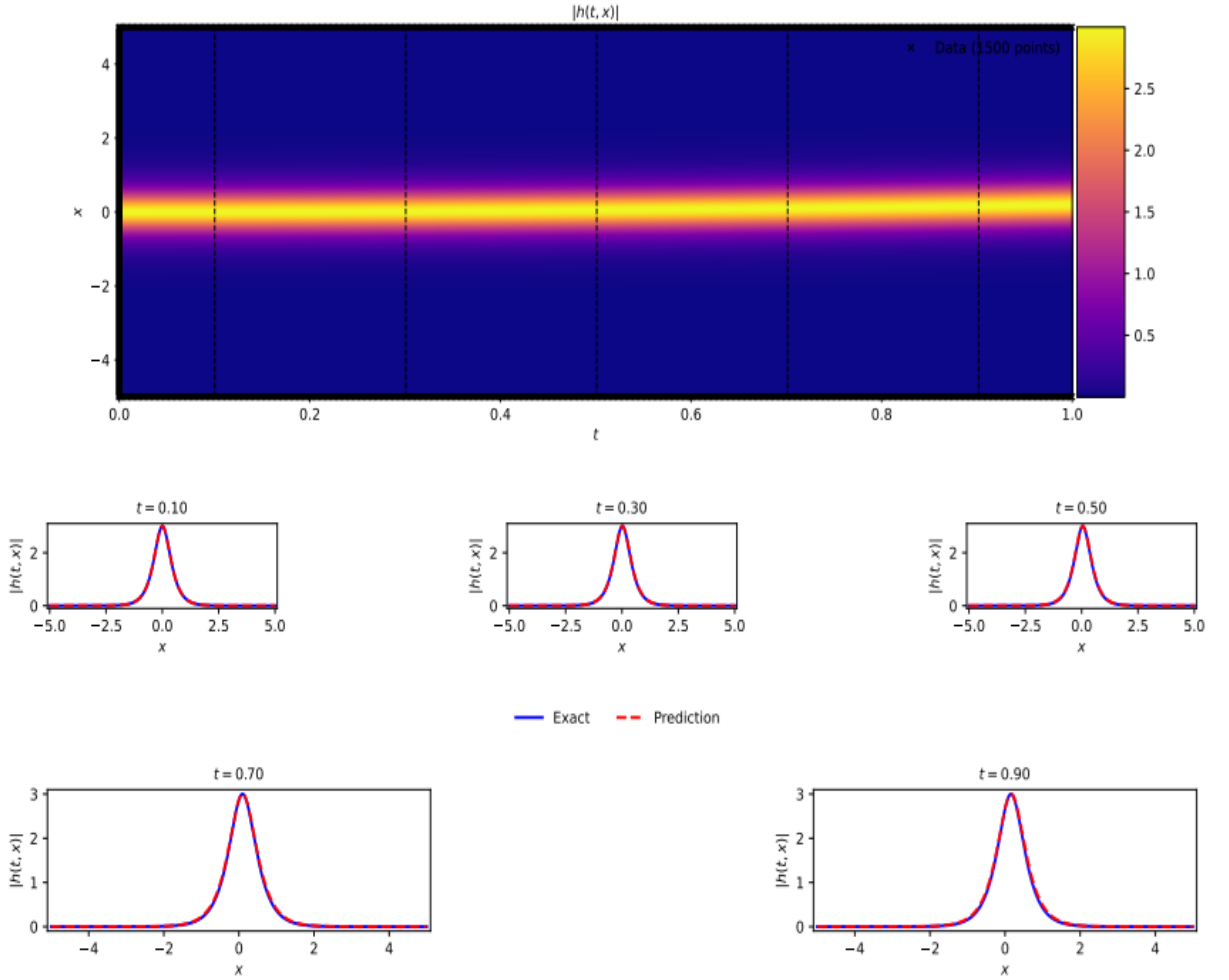


Figura 4.14: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 20542s (aprox. 5.7hrs)

Solución con solo potencial y velocidad (con parametro $c_0 = 0,05$)

Esta solución es la misma que (4.1) con la característica de que el soliton solo esta bajo la influencia del potencial lineal y una velocidad en su dinamica su grafica es la figura 4.21

La PINN obtuvo lo siguiente en las figuras 4.22, 4.23 y 4.24

En esta solución se puede observar que la red neuronal empieza a tener

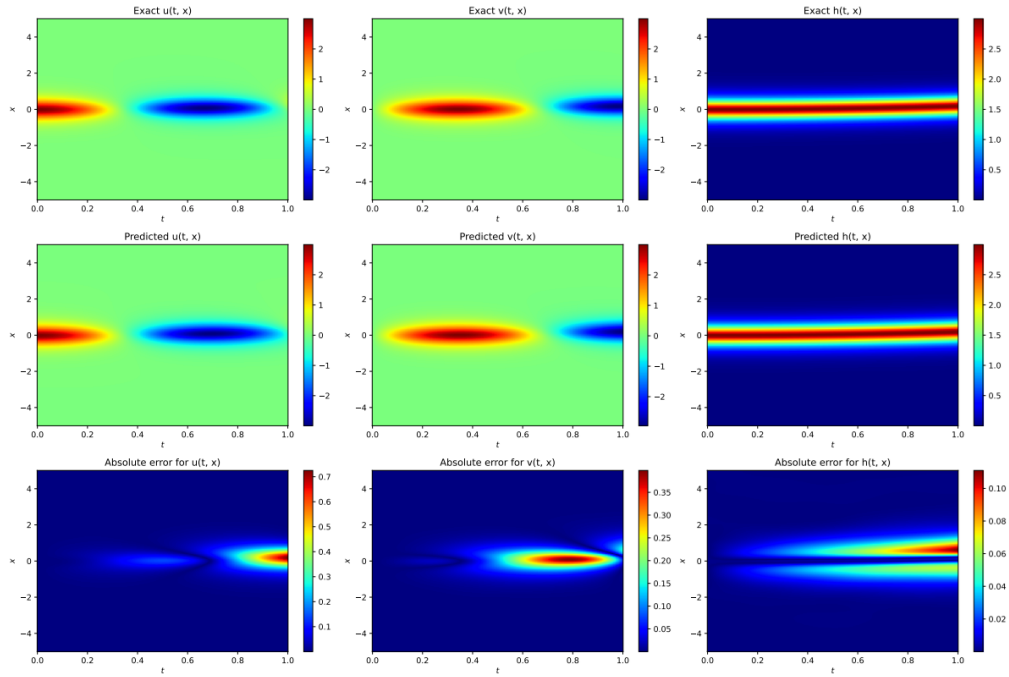


Figura 4.15: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u: 1.234977e-01, Error v: 9.751625e-02, Error h: 2.321947e-02

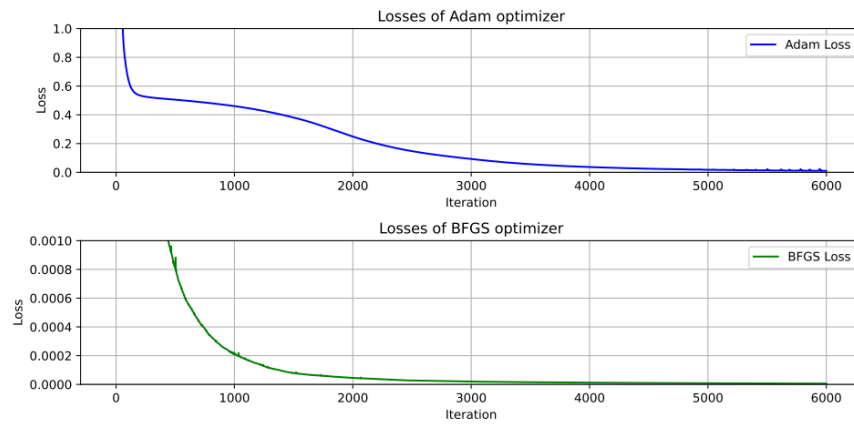


Figura 4.16: grafica del error de los optimizadores por iteración

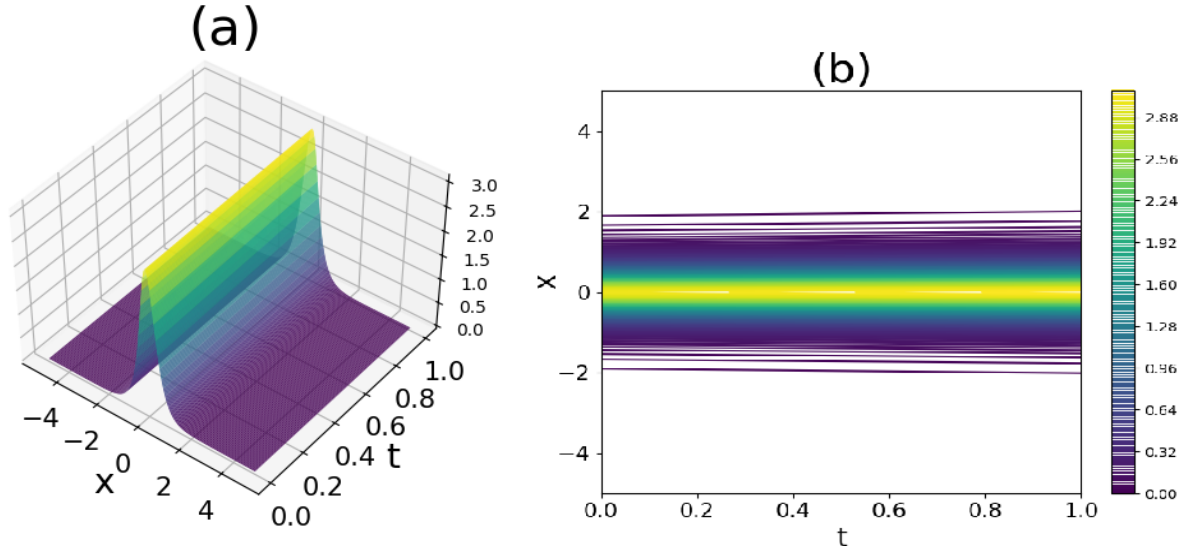


Figura 4.17: Dinamica de 1 soliton con los parametros $c_0 = 0.05$, $\lambda_0 = 0$, $\eta_{01} = 1.5$, $\kappa_{01} = 0$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

problemas en el aprendizaje, esto es debido a que el potencial lineal mueve el soliton en la dirección contraria que la velocidad por lo que a travez de la dinamica avanza la PINN tiene problemas en la predicción de la complejidad de la dinamica.

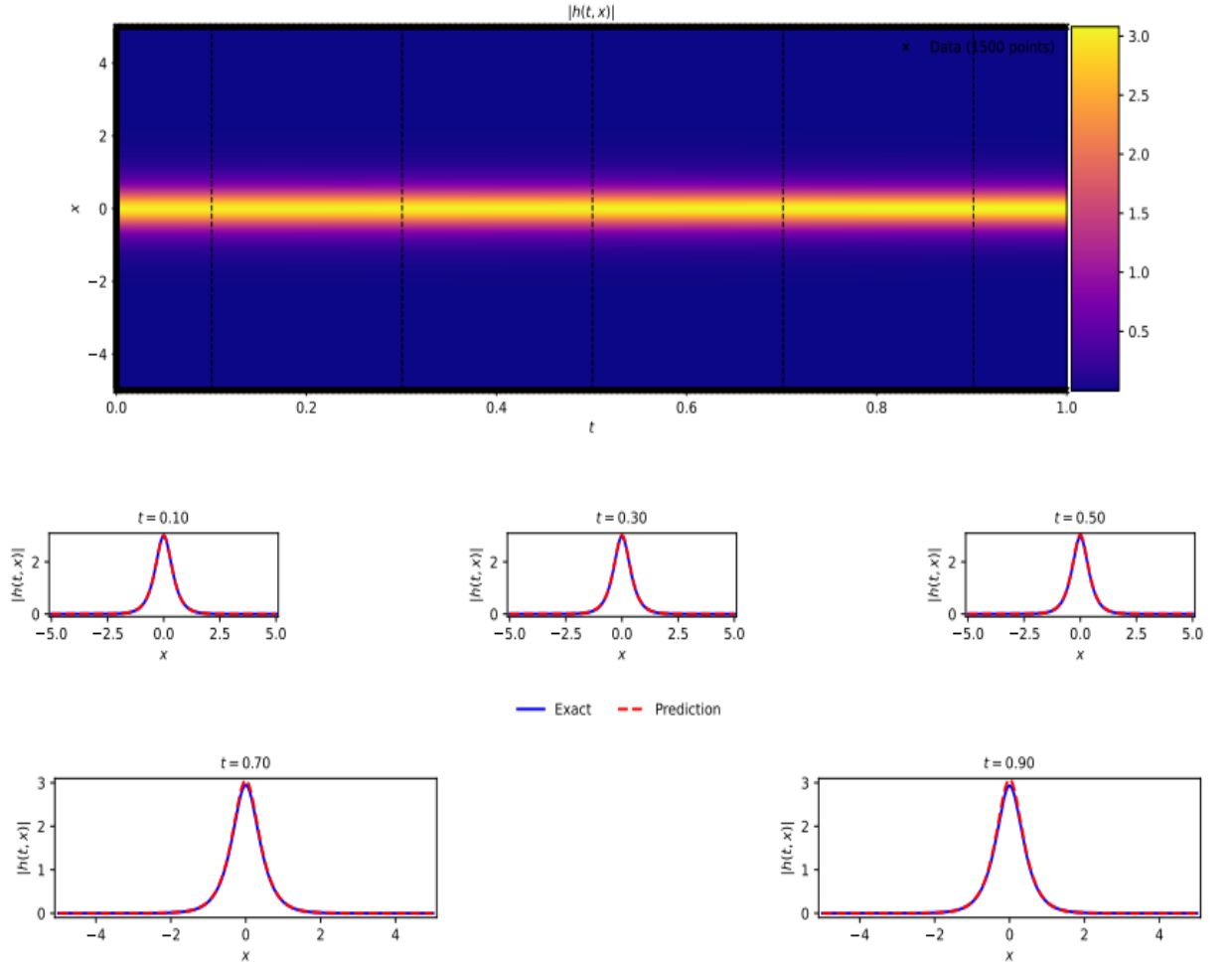


Figura 4.18: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 22568s (aprox. 6.2hrs)

Solución con solo potencial y dispersión (con parametro $c_0 = 0,05$)

Esta solución es la misma que (4.2) con la característica de que el soliton es influenciado por el potencial lineal además de la dispersión no constante su grafica es la figura 4.25

La PINN obtuvo lo siguiente en las figuras 4.26, 4.27 y 4.28

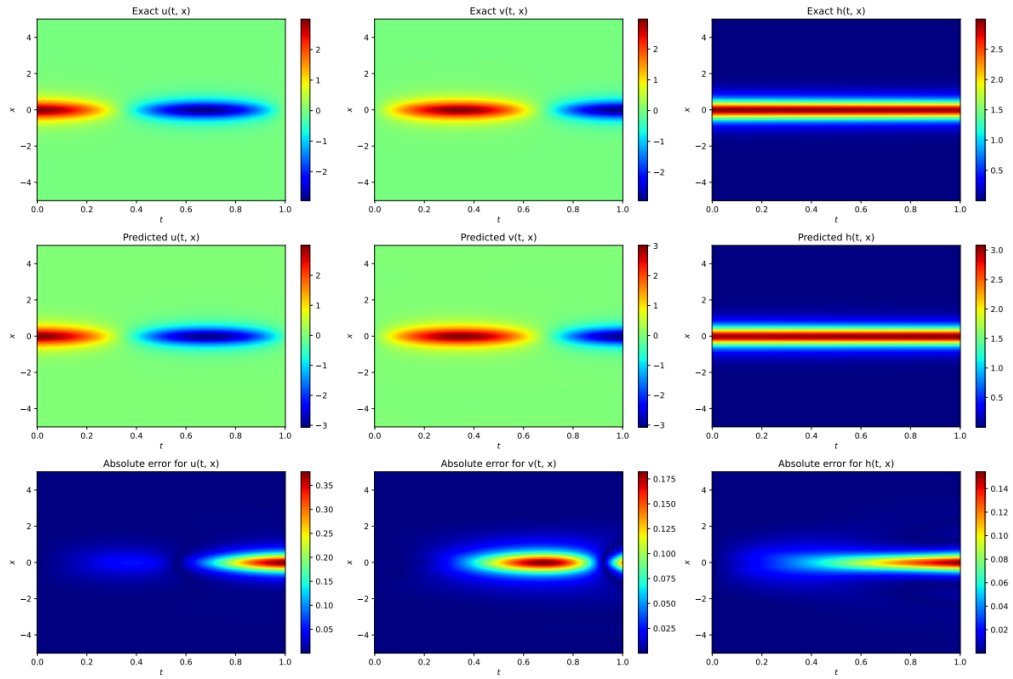


Figura 4.19: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u: 7.746521e-02, Error v: 5.313268e-02, Error h: 2.906419e-02

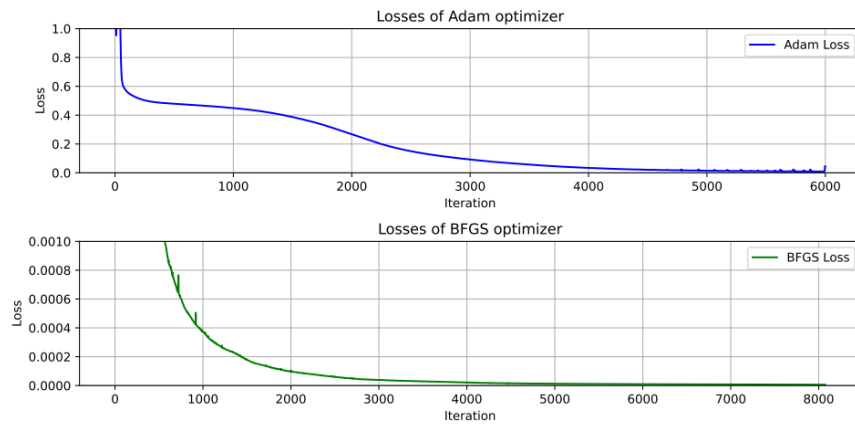


Figura 4.20: grafica del error de los optimizadores por iteración

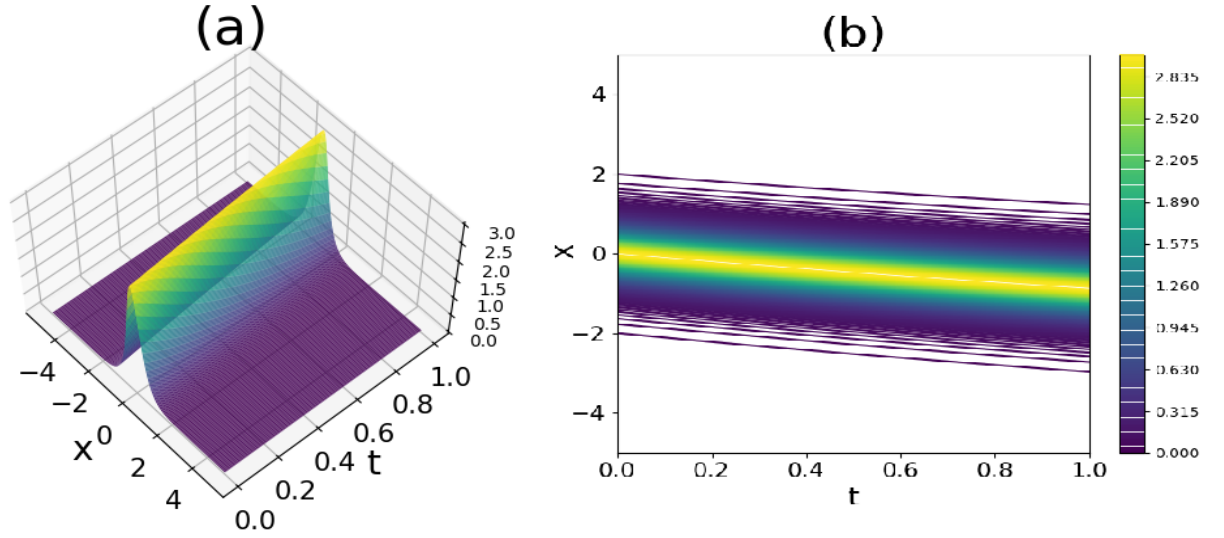


Figura 4.21: Dinamica de 1 soliton con los parametros $c_0 = 0.05$, $\lambda_0 = -.2$. $\eta_{01} = 1.5$, $\kappa_{01} = .5$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

Solución con solo potencial, velocidad y dispersión (con parametro $c_0 = 0,05$)

Esta solución es la misma que (4.2) con la característica de que el soliton es influenciado por el potencial lineal además de la dispersion no constante y una velocidad en su dinamica su grafica es la figura 4.29.

La PINN obtuvo lo siguiente en las figuras 4.30, 4.31 y 4.32.

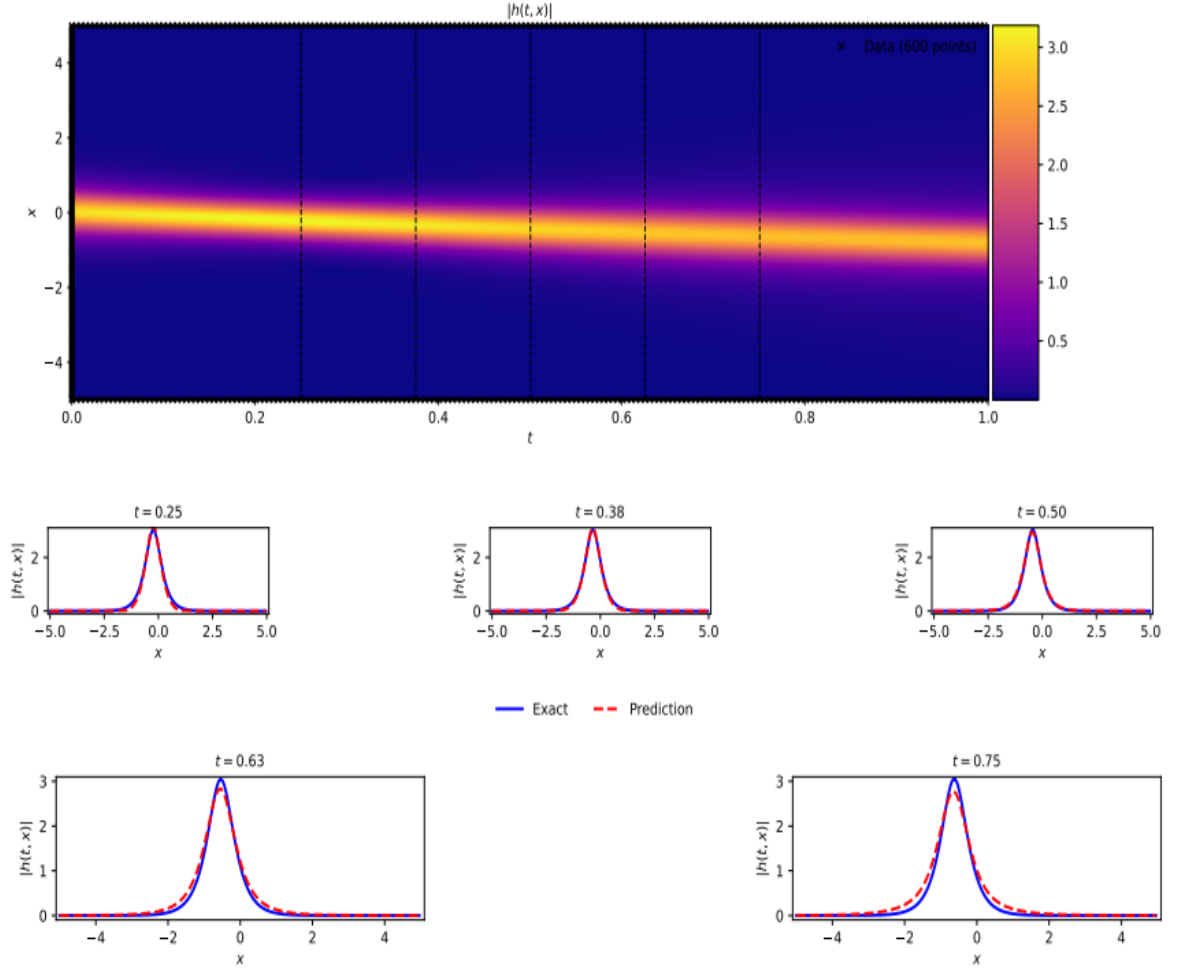


Figura 4.22: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 19549s (aprox. 5.4hrs)

4.2. Análisis General

Todas las soluciones presentadas en esta sección fueron estudiadas usando una red neuronal con la siguiente arquitectura: $[2, 100, 100, 100, 100, 2]$, es decir, una red con cuatro capas ocultas de 100 neuronas cada una. El dominio considerado fue $x \in [-5, 5]$, $t \in [0, 1]$, y se utilizaron el mismo número de datos de muestreo en cada caso. La motivación fue estudiar cómo distintos parámetros afectan la capacidad de aprendizaje de la PINN.

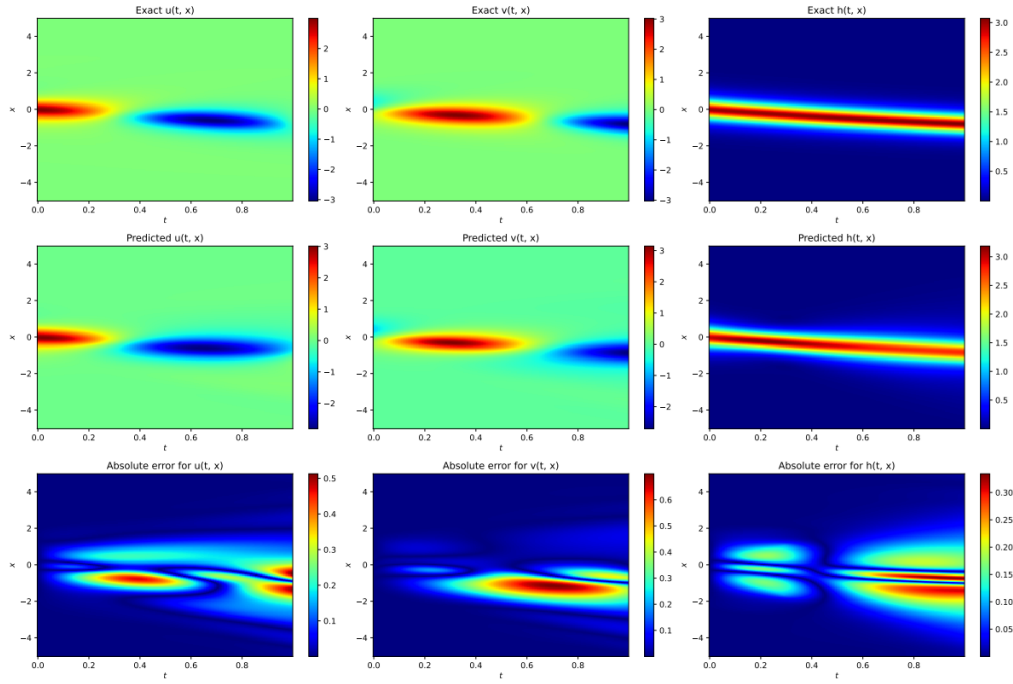


Figura 4.23: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u: 1.906296e-01, Error v: 2.663305e-01, Error h: 1.032756e-01

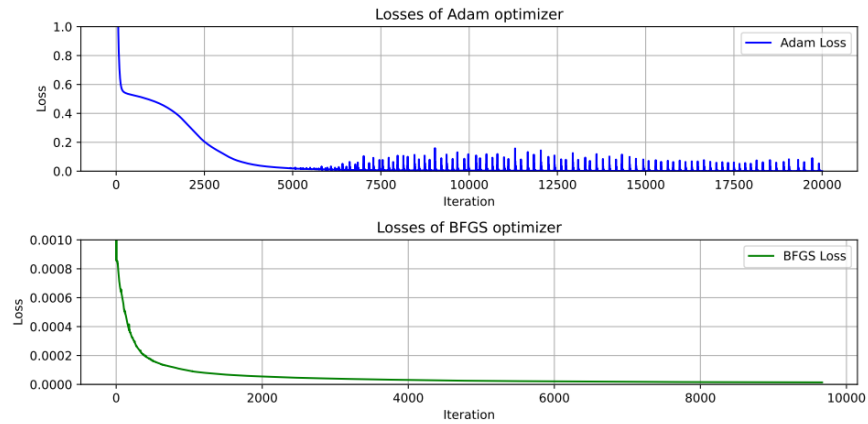


Figura 4.24: grafica del error de los optimizadores por iteración

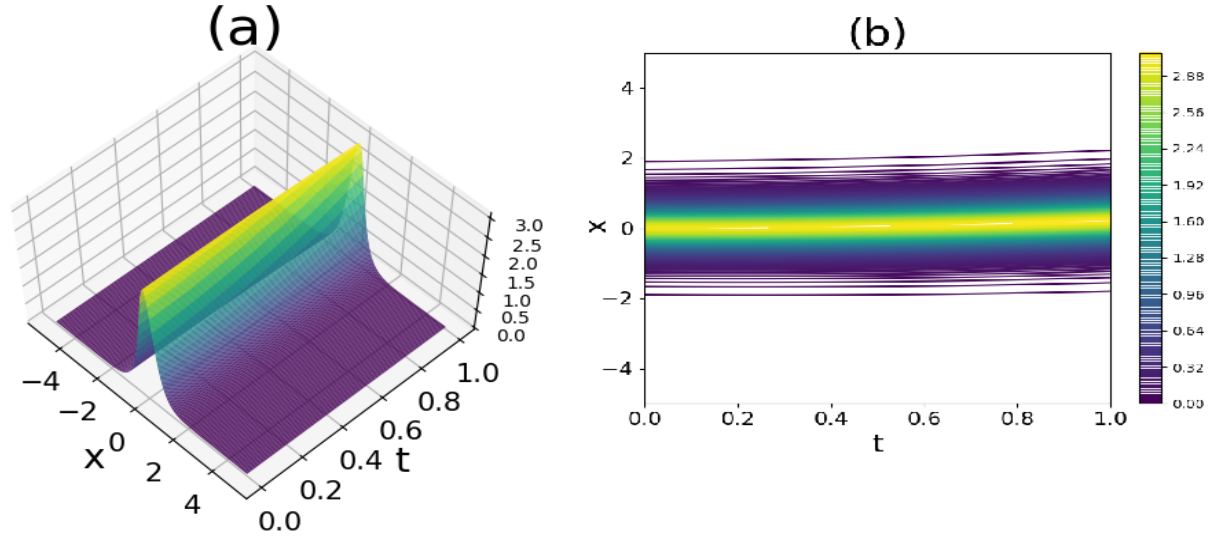


Figura 4.25: Dinamica de 1 soliton con los parametros $c_0 = 0.05$, $\lambda_0 = -.2$. $\eta_{01} = 1.5$, $\kappa_{01} = 0$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

4.3. Comparación de Resultados

A continuación, se presentan los resultados obtenidos para cada una de las configuraciones estudiadas (Cuadro 4.1), poniendo énfasis en los errores alcanzados, los tiempos de entrenamiento y los parámetros clave de cada solución.

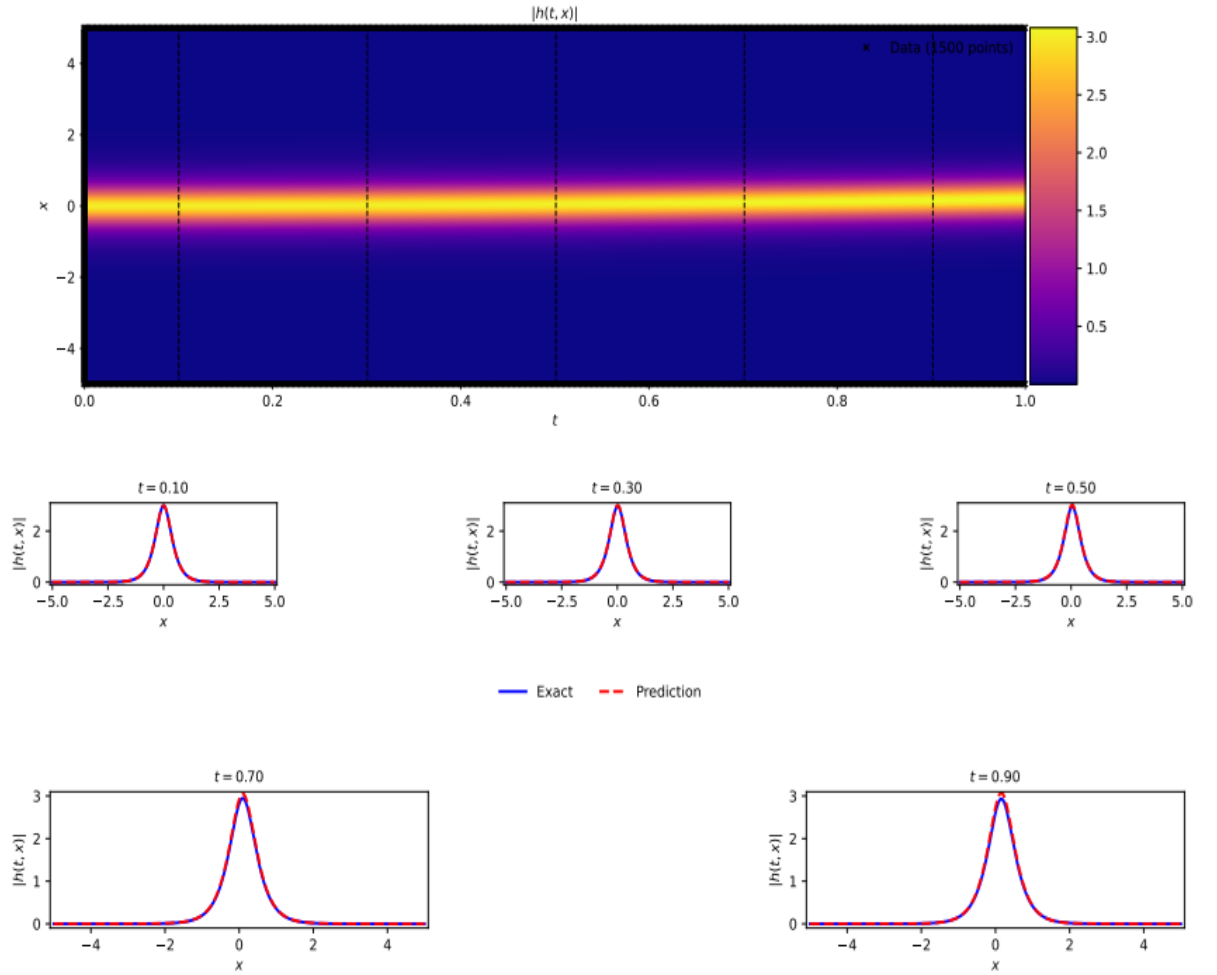


Figura 4.26: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 19830s (aprox. 5.5hrs)

4.4. Discusión de Resultados

- La solución más simple (caso 2) presenta uno de los menores errores absolutos y es aprendida con facilidad por la red, a pesar de que el tiempo de entrenamiento es elevado. Esto evidencia que la complejidad de la dinámica juega un papel más importante que el tiempo en la convergencia de la red.

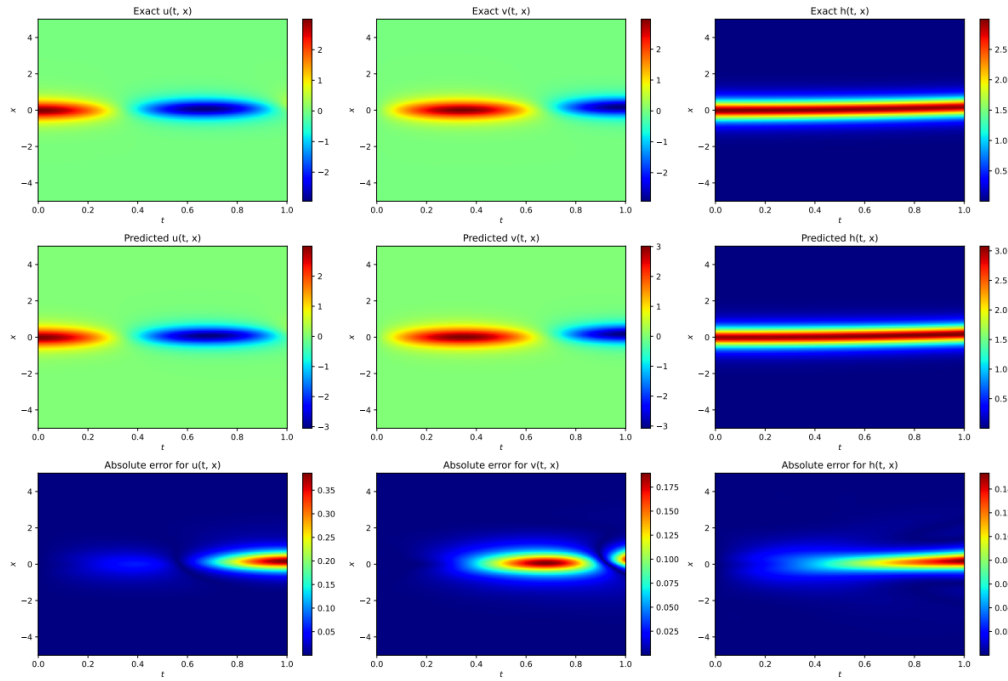


Figura 4.27: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u: $8.017419e-02$, Error v: $5.488514e-02$, Error h: $2.889751e-02$

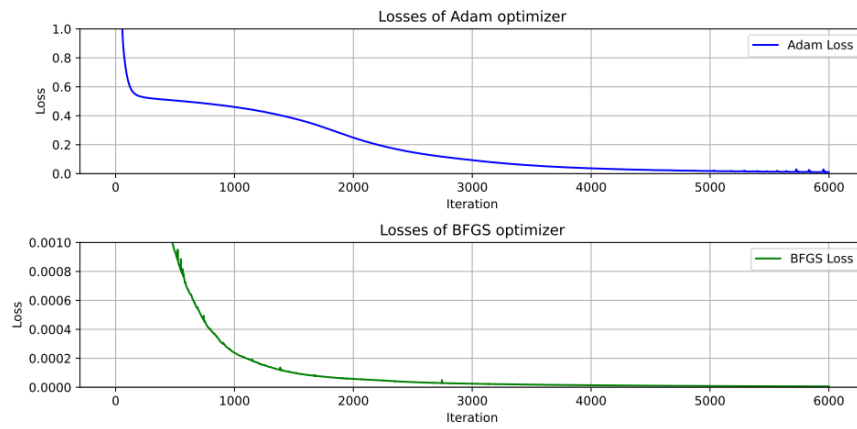


Figura 4.28: grafica del error de los optimizadores por iteración

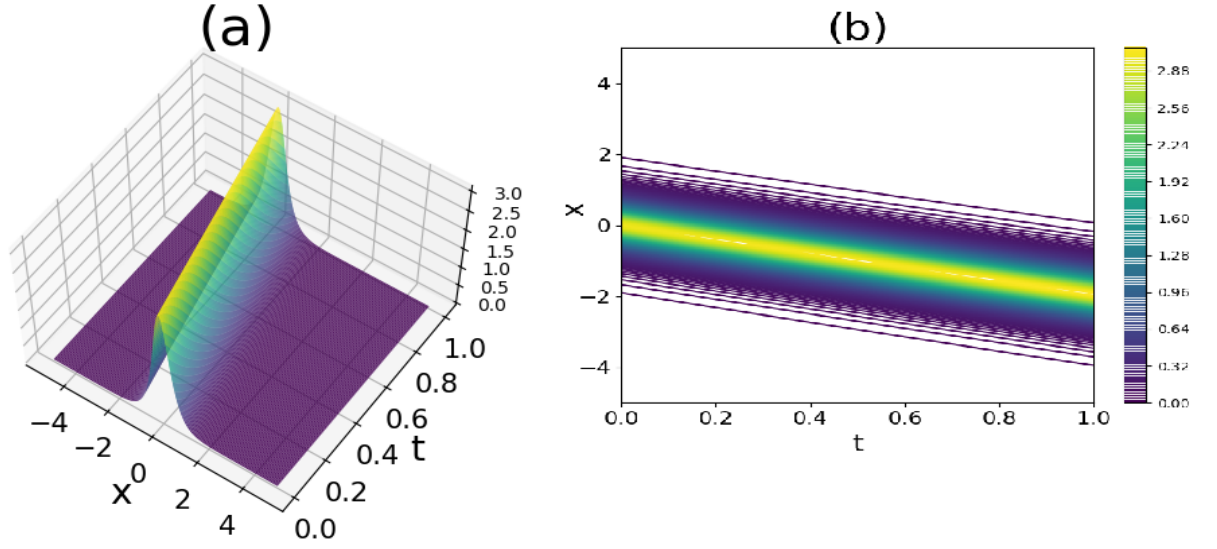


Figura 4.29: Dinamica de 1 soliton con los parametros $c_0 = 0.05$, $\lambda_0 = -.2$. $\eta_{01} = 1.5$, $\kappa_{01} = 1$. (a) muestra el comportamiento espacio-temporal; y (b) el mapa de contorno correspondiente.

Caso	Error u	Error v	Error h	Tiempo (s)
Solitón general	0.1913	0.2663	0.1045	12291
Solución simple	0.1193	0.0958	0.0227	21825
Solo velocidad	0.3953	0.3959	0.0501	25155
Solo potencial	0.1235	0.0975	0.0232	20542
Solo dispersión	0.0775	0.0531	0.0291	22568
Potencial + velocidad	0.1906	0.2663	0.1033	19549
Potencial + dispersión	0.0802	0.0549	0.0289	19830
Potencial + velocidad + dispersión	0.0791	0.0865	0.0289	22225

Cuadro 4.1: Comparación de los errores obtenidos por la PINN en diferentes configuraciones del solitón.

- El caso más complejo (caso 1), con todos los efectos presentes, muestra que la red tiene dificultad para reducir el error por debajo de 1×10^{-2} , especialmente en la fase inicial de entrenamiento, incluso tras 10000 iteraciones del optimizador Adam.

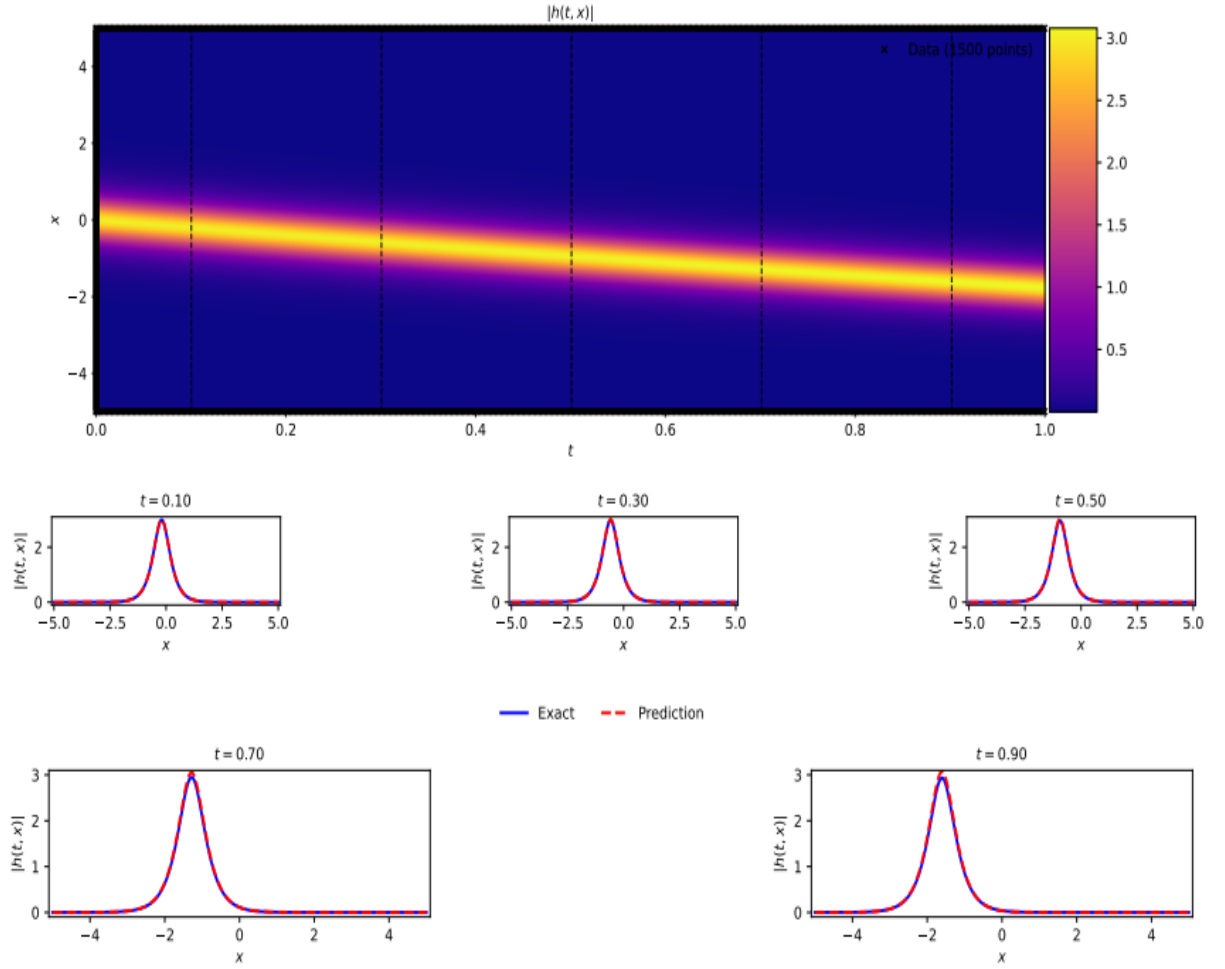


Figura 4.30: Imagen que muestra la grafica predicha por la red neuronal el tiempo de entrenamiento fue de 22225s (aprox. 6.1hrs)

- El caso con solo velocidad (caso 3) presenta el mayor error en u y v , lo que sugiere que la introducción de velocidad complica significativamente el aprendizaje.
- Los casos con solo potencial o solo dispersión (casos 4 y 5) arrojan mejores resultados que el caso general, lo cual sugiere que el efecto combinado de los parámetros incrementa la complejidad para la red.

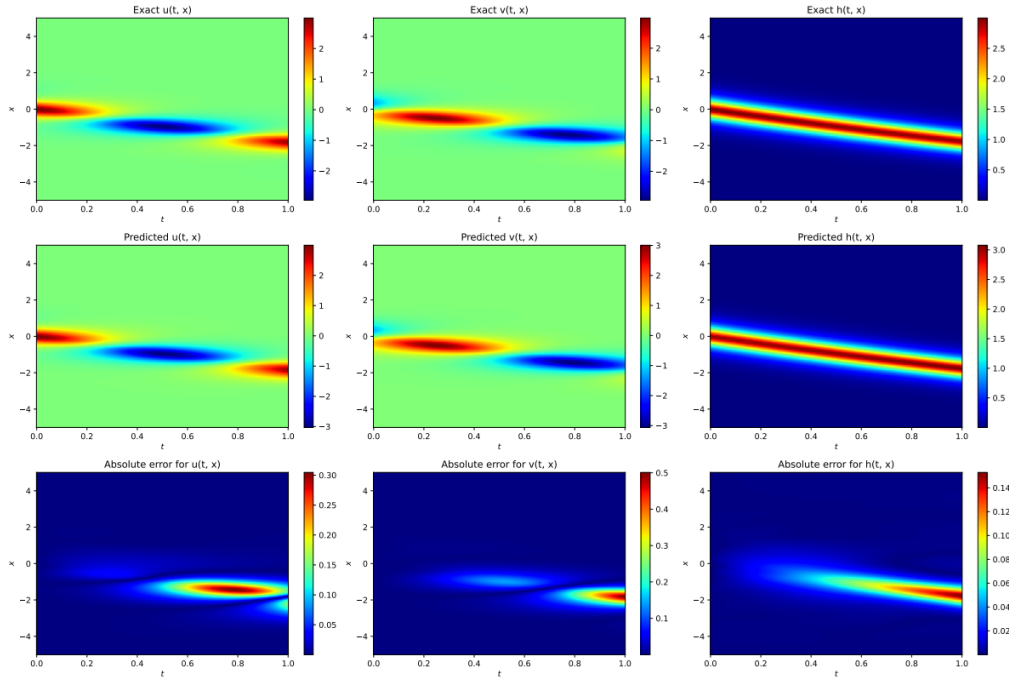


Figura 4.31: Comparacion de las partes real, imaginaria y la funcion total y su error absoluto respectivamente Error u: $7.907374e-02$, Error v: $8.651064e-02$, Error h: $2.885101e-02$

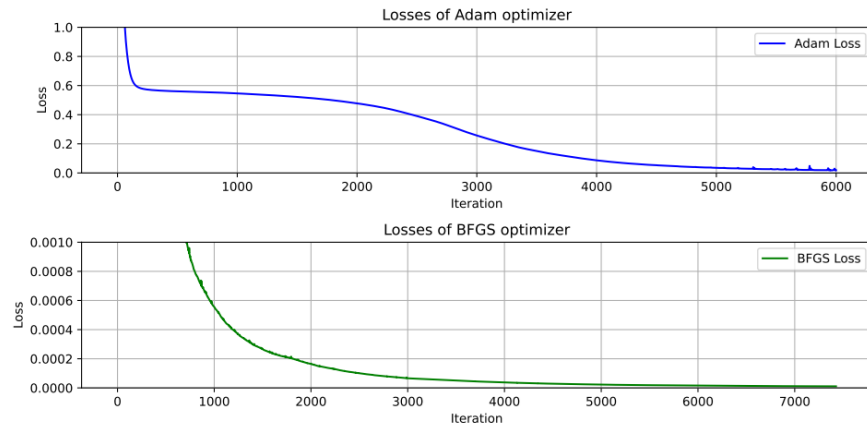


Figura 4.32: grafica del error de los optimizadores por iteración

- En el caso 6, la combinación de potencial y velocidad parece generar conflictos en la dinámica, pues el potencial y la velocidad inducen movimientos en direcciones opuestas, dificultando la predicción.
- Finalmente, en los casos 7 y 8, a pesar de la mayor complejidad, la red logra obtener errores relativamente bajos, comparables al caso 5, lo que puede indicar que algunos efectos como la dispersión suavizan la solución en el dominio y permiten un mejor aprendizaje.

4.5. Conclusiones Parciales

El análisis demuestra que la arquitectura de la red neuronal utilizada es capaz de aprender soluciones de distinta complejidad, aunque con un costo computacional importante. La presencia de velocidad y potencial lineal simultáneamente representa uno de los mayores desafíos para el entrenamiento. La dispersión, por otro lado, no parece afectar negativamente el aprendizaje e incluso puede ayudar a estabilizar la dinámica de la solución. Estos resultados son clave para futuros diseños de experimentos numéricos usando PINNs en ecuaciones de Schrodinger no lineales con potencial.

Capítulo 5

Conclusiones

El trabajo realizado durante esta investigación de maestría demostró la eficacia de las redes neuronales de aprendizaje profundo, específicamente las PINNs (Physics-Informed Neural Networks), para abordar la solución de la ecuación no lineal de Schrödinger con potencial lineal. A través de un proceso iterativo y ajustes en los parámetros de la red, se logró reducir significativamente el error de aprendizaje, acercándose a valores aceptables. Esto resalta la complejidad inherente de los problemas no lineales y la capacidad de las PINNs para adaptarse a dinámicas desafiantes, como las presentes en la dinámica de solitones.

Los resultados obtenidos, respaldados por las gráficas y análisis presentados, evidencian que factores como el término chirpeado, la velocidad, el potencial y la dispersión tienen un impacto crítico en el aprendizaje de la red. La optimización de los hiperparámetros, como el número de puntos de datos y las iteraciones de los optimizadores (Adam y L-BFGS-B), fue clave para mejorar la precisión del modelo. Además, se confirmó que incrementar el tamaño de la red no siempre conduce a mejores resultados, sino que puede aumentar innecesariamente el tiempo de entrenamiento.

En conclusión, este proyecto no solo cumplió con el objetivo de reducir el error, sino que también proporcionó valiosas lecciones sobre el uso de redes neuronales en problemas de física matemática. Los hallazgos serán fundamentales para futuras investigaciones y aplicaciones en el campo de las ecuaciones diferenciales no lineales.

Apéndice A

Programa PINN

```
1 import tensorflow.compat.v1 as tf
2 tf.disable_eager_execution()
3 import numpy as np
4 import matplotlib
5 matplotlib.use('pdf')
6 import matplotlib.pyplot as plt
7 import scipy.io
8 from scipy.interpolate import griddata
9 from pyDOE import lhs
10 import time
11 import matplotlib.gridspec as gridspec
12 from mpl_toolkits.axes_grid1 import make_axes_locatable
13 import deepxde as dde
14
15 # Set the random seed for NumPy
16 np.random.seed(1234)
17
18 # Set the random seed for TensorFlow
19 tf.set_random_seed(1234)
20
21 print("GPU Available: ", tf.test.is_gpu_available())
22
23
24
25 class PhysicsInformedNN:
26     # Initialize the class
27     def __init__(self, x0, u0, v0, tb, X_f, layers, lb, ub):
28
29         X0 = np.concatenate((x0, 0 * x0), 1) # (x0, 0)
30         X_lb = np.concatenate((0 * tb + lb[0], tb), 1) # (lb[0], tb)
31         X_ub = np.concatenate((0 * tb + ub[0], tb), 1) # (ub[0], tb)
32
33         self.lb = lb
34         self.ub = ub
35
36         self.x0 = X0[:, 0:1]
37         self.t0 = X0[:, 1:2]
```

```

38
39     self.x_lb = X_lb[:, 0:1]
40     self.t_lb = X_lb[:, 1:2]
41
42     self.x_ub = X_ub[:, 0:1]
43     self.t_ub = X_ub[:, 1:2]
44
45     self.x_f = X_f[:, 0:1]
46     self.t_f = X_f[:, 1:2]
47
48     self.u0 = u0
49     self.v0 = v0
50
51     # Initialize NNs
52     self.layers = layers
53     self.weights, self.biases = self.initialize_NN(layers)
54
55     # tf Placeholders
56     self.x0_tf = tf.placeholder(tf.float32, shape=[None, self.x0.shape
57                                [1]])
58     self.t0_tf = tf.placeholder(tf.float32, shape=[None, self.t0.shape
59                                [1]])
60
61     self.u0_tf = tf.placeholder(tf.float32, shape=[None, self.u0.shape
62                                [1]])
63     self.v0_tf = tf.placeholder(tf.float32, shape=[None, self.v0.shape
64                                [1]])
65
66     self.x_lb_tf = tf.placeholder(tf.float32, shape=[None, self.x_lb.
67                                shape[1]])
68     self.t_lb_tf = tf.placeholder(tf.float32, shape=[None, self.t_lb.
69                                shape[1]])
70
71     self.x_ub_tf = tf.placeholder(tf.float32, shape=[None, self.x_ub.
72                                shape[1]])
73     self.t_ub_tf = tf.placeholder(tf.float32, shape=[None, self.t_ub.
74                                shape[1]])
75
76     self.x_f_tf = tf.placeholder(tf.float32, shape=[None, self.x_f.
77                                shape[1]])
78     self.t_f_tf = tf.placeholder(tf.float32, shape=[None, self.t_f.
79                                shape[1]])
80
81     # tf Graphs
82     self.u0_pred, self.v0_pred, _, _ = self.net_uv(self.x0_tf, self.
83                                t0_tf)
84     self.u_lb_pred, self.v_lb_pred, self.u_x_lb_pred, self.v_x_lb_pred
85     = self.net_uv(self.x_lb_tf, self.t_lb_tf)
86     self.u_ub_pred, self.v_ub_pred, self.u_x_ub_pred, self.v_x_ub_pred
87     = self.net_uv(self.x_ub_tf, self.t_ub_tf)
88     self.f_u_pred, self.f_v_pred = self.net_f_uv(self.x_f_tf, self.
89                                t_f_tf)
90
91     # Loss
92     self.loss = tf.reduce_mean(tf.square(self.u0_tf - self.u0_pred)) +
93     \

```

```

79         tf.reduce_mean(tf.square(self.v0_tf - self.v0_pred)) + \
80         tf.reduce_mean(tf.square(self.u_lb_pred - self.
81             u_ub_pred)) + \
82         tf.reduce_mean(tf.square(self.v_lb_pred - self.
83             v_ub_pred)) + \
84         tf.reduce_mean(tf.square(self.u_x_lb_pred - self.
85             u_x_ub_pred)) + \
86         tf.reduce_mean(tf.square(self.v_x_lb_pred - self.
87             v_x_ub_pred)) + \
88         tf.reduce_mean(tf.square(self.f_u_pred)) + \
89         tf.reduce_mean(tf.square(self.f_v_pred))
90
91     # Optimizers using TensorFlow 2.x
92     self.optimizer = dde.optimizers.tensorflow_compat_v1.
93         scipy_optimizer.ScipyOptimizerInterface(
94             self.loss, method='L-BFGS-B',
95             options={'maxiter': 10000, 'maxfun': 10000, 'maxcor': 50, '
96                 maxls': 50, 'ftol': 1.0 * np.finfo(float).eps})
97
98     #####
99     self.optimizer_Adam = tf.train.AdamOptimizer()
100     self.train_op_Adam = self.optimizer_Adam.minimize(self.loss)
101     self.gradients = tf.gradients(self.loss, self.weights)
102
103     # tf session
104     self.sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=
105         True,
106                                     log_device_placement=
107                                     True))
108
109     init = tf.global_variables_initializer()
110     self.sess.run(init)
111
112     self.losses_adam = []
113     self.losses_bfgs = []
114
115     def initialize_NN(self, layers):
116         weights = []
117         biases = []
118         num_layers = len(layers)
119         for l in range(0, num_layers - 1):
120             W = tf.Variable(tf.random_normal([layers[l], layers[l + 1]],
121                 stddev=np.sqrt(1 / layers[l])),
122                 dtype=tf.float32)
123             b = tf.Variable(tf.random_normal([1, layers[l + 1]]), dtype=tf.
124                 float32)
125             weights.append(W)
126             biases.append(b)
127         return weights, biases
128
129     def neural_net(self, X, weights, biases):
130         num_layers = len(weights) + 1
131
132         H = 2.0 * (X - self.lb) / (self.ub - self.lb) - 1.0
133         for l in range(0, num_layers - 2):
134             W = weights[l]

```

```

125         b = biases[l]
126         H = tf.tanh(tf.add(tf.matmul(H, W), b))
127         W = weights[-1]
128         b = biases[-1]
129         Y = tf.add(tf.matmul(H, W), b)
130         return Y
131
132     def net_uv(self, x, t):
133         X = tf.concat([x, t], 1)
134
135         uv = self.neural_net(X, self.weights, self.biases)
136         u = uv[:, 0:1]
137         v = uv[:, 1:2]
138
139         u_x = tf.gradients(u, x)[0]
140         v_x = tf.gradients(v, x)[0]
141
142         return u, v, u_x, v_x
143
144     def net_f_uv(self, x, t):
145         u, v, u_x, v_x = self.net_uv(x, t)
146
147         u_t = tf.gradients(u, t)[0]
148         u_xx = tf.gradients(u_x, x)[0]
149
150         v_t = tf.gradients(v, t)[0]
151         v_xx = tf.gradients(v_x, x)[0]
152
153         f_v = -v_t+.5*tf.exp(-.05 * t)*u_xx+1*(u**2+v**2)*u-2*(-.2)*x*u
154         f_u = u_t+.5*tf.exp(-.05 * t)*v_xx+1*(u**2+v**2)*v-2*(-.2)*x*v
155
156         return f_u, f_v
157
158     def callback(self, loss):
159         print('Loss L-BFGS-B:', loss)
160         self.losses_bfgs.append(loss)
161
162     def train(self, nIter):
163
164         tf_dict = {self.x0_tf: self.x0, self.t0_tf: self.t0,
165                   self.u0_tf: self.u0, self.v0_tf: self.v0,
166                   self.x_lb_tf: self.x_lb, self.t_lb_tf: self.t_lb,
167                   self.x_ub_tf: self.x_ub, self.t_ub_tf: self.t_ub,
168                   self.x_f_tf: self.x_f, self.t_f_tf: self.t_f}
169
170         start_time = time.time()
171         for it in range(nIter):
172             _, loss_value, gradients = self.sess.run([self.train_op_Adam,
173                                                       self.loss, self.gradients], tf_dict)
174             self.losses_adam.append(loss_value)
175
176             # Print
177             if it \% 100 == 0:
178                 elapsed = time.time() - start_time
179                 print(f'Iteration {it}, Loss Adam: {loss_value}')
180                 start_time = time.time()

```

```

181         self.optimizer.minimize(self.sess, feed_dict=tf_dict, fetches=[self
182             .loss], loss_callback=self.callback)
183
184     def predict(self, X_star):
185
186         tf_dict = {self.x0_tf: X_star[:, 0:1], self.t0_tf: X_star[:,
187             1:2]}
188         u_star = self.sess.run(self.u0_pred, tf_dict)
189         v_star = self.sess.run(self.v0_pred, tf_dict)
190
191         tf_dict = {self.x_f_tf: X_star[:, 0:1], self.t_f_tf: X_star[:,
192             1:2]}
193         f_u_star = self.sess.run(self.f_u_pred, tf_dict)
194         f_v_star = self.sess.run(self.f_v_pred, tf_dict)
195
196         return u_star, v_star, f_u_star, f_v_star
197
198 if __name__ == "__main__":
199     noise = 0.0
200
201     # Domain bounds
202     lb = np.array([-5, 0])
203     ub = np.array([5, 1])
204
205     N0 = 500
206     N_b = 500
207     N_f = 50000
208     layers = [2, 100, 100, 100, 100, 2]
209
210     data = scipy.io.loadmat('C:.....
211         NLSE_potencial_lineal_isoliton.mat')
212
213     t = data['t_values'].flatten()[:, None]
214     x = data['x_values'].flatten()[:, None]
215     Exact = data['q_values']
216     Exact_u = np.real(Exact)
217     Exact_v = np.imag(Exact)
218     Exact_h = np.sqrt(Exact_u ** 2 + Exact_v ** 2)
219
220     X, T = np.meshgrid(x, t)
221
222     X_star = np.hstack((X.flatten()[:, None], T.flatten()[:, None]))
223     u_star = Exact_u.T.flatten()[:, None]
224     v_star = Exact_v.T.flatten()[:, None]
225     h_star = Exact_h.T.flatten()[:, None]
226
227     idx_x = np.random.choice(x.shape[0], N0, replace=False)
228     x0 = x[idx_x, :]
229     u0 = Exact_u[idx_x, 0:1]
230     v0 = Exact_v[idx_x, 0:1]
231
232     idx_t = np.random.choice(t.shape[0], N_b, replace=False)
233     tb = t[idx_t, :]
234
235     X_f = lb + (ub - lb) * lhs(2, N_f)

```

```

234 model = PhysicsInformedNN(x0, u0, v0, tb, X_f, layers, lb, ub)
235
236 start_time = time.time()
237
238 model.train(6000)
239
240 elapsed = time.time() - start_time
241 print('Training time: \%.4f' \% (elapsed))
242
243 u_pred, v_pred, f_u_pred, f_v_pred = model.predict(X_star)
244 h_pred = np.sqrt(u_pred ** 2 + v_pred ** 2)
245
246 error_u = np.linalg.norm(u_star - u_pred, 2) / np.linalg.norm(u_star,
2)
247 error_v = np.linalg.norm(v_star - v_pred, 2) / np.linalg.norm(v_star,
2)
248 error_h = np.linalg.norm(h_star - h_pred, 2) / np.linalg.norm(h_star,
2)
249 print('Error u: \%.e' \% (error_u))
250 print('Error v: \%.e' \% (error_v))
251 print('Error h: \%.e' \% (error_h))
252
253 # Verify shapes and dimensions of input arrays
254 print("Shapes:")
255 print("X_star:", X_star.shape)
256 print("u_pred:", u_pred.shape)
257 print("v_pred:", v_pred.shape)
258 print("h_pred:", h_pred.shape)
259 print("f_u_pred:", f_u_pred.shape)
260 print("f_v_pred:", f_v_pred.shape)
261 print("X:", X.shape)
262 print("T:", T.shape)
263
264 U_pred = griddata(X_star, u_pred.flatten(), (X, T), method='cubic')
265 V_pred = griddata(X_star, v_pred.flatten(), (X, T), method='cubic')
266 H_pred = griddata(X_star, h_pred.flatten(), (X, T), method='cubic')
267 U_star = griddata(X_star, u_star.flatten(), (X, T), method='cubic')
268 V_star = griddata(X_star, v_star.flatten(), (X, T), method='cubic')
269 H_star = griddata(X_star, h_star.flatten(), (X, T), method='cubic')
270
271 FU_pred = griddata(X_star, f_u_pred.flatten(), (X, T), method='cubic')
272 FV_pred = griddata(X_star, f_v_pred.flatten(), (X, T), method='cubic')
273
274 #####
275 ##### Plotting #####
276 #####
277 def newfig(width, height):
278     fig = plt.figure(figsize=(width, height))
279     return fig, fig.gca()
280
281 X0 = np.concatenate((x0, 0 * x0), 1) # (x0, 0)
282 X_lb = np.concatenate((0 * tb + lb[0], tb), 1) # (lb[0], tb)
283 X_ub = np.concatenate((0 * tb + ub[0], tb), 1) # (ub[0], tb)
284 X_u_train = np.vstack([X0, X_lb, X_ub])
285
286 fig, ax = newfig(15.0, 12.0)
287 ax.axis('off')

```

```

288
289 ##### Row 0: h(t,x) #####
290 gs0 = gridspec.GridSpec(1, 2)
291 gs0.update(top=1 - 0.06, bottom=1 - 1 / 3, left=0.15, right=0.85,
292            wspace=0)
293 ax = plt.subplot(gs0[:, :])
294 h = ax.imshow(H_pred.T, interpolation='nearest', cmap='plasma',
295               extent=[lb[1], ub[1], lb[0], ub[0]],
296               origin='lower', aspect='auto')
297 divider = make_axes_locatable(ax)
298 cax = divider.append_axes("right", size="5%", pad=0.05)
299 fig.colorbar(h, cax=cax)
300
301 ax.plot(X_u_train[:, 1], X_u_train[:, 0], 'kx', label='Data (%d points
302         )' % (X_u_train.shape[0]), markersize=4,
303         clip_on=False)
304
305 line = np.linspace(x.min(), x.max(), 2)[: , None]
306 ax.plot(t[50] * np.ones((2, 1)), line, 'k--', linewidth=1)
307 ax.plot(t[150] * np.ones((2, 1)), line, 'k--', linewidth=1)
308 ax.plot(t[250] * np.ones((2, 1)), line, 'k--', linewidth=1)
309 ax.plot(t[350] * np.ones((2, 1)), line, 'k--', linewidth=1)
310 ax.plot(t[450] * np.ones((2, 1)), line, 'k--', linewidth=1)
311
312 ax.set_xlabel('$t$')
313 ax.set_ylabel('$x$')
314 leg = ax.legend(frameon=False, loc='best')
315 # plt.setp(leg.get_texts(), color='w')
316 ax.set_title('$|h(t,x)|$', fontsize=10)
317
318 ##### Row 1: h(t,x) slices #####
319 gs1 = gridspec.GridSpec(1, 3)
320 gs1.update(top=1 - 1 / 3, bottom=0, left=0.1, right=0.9, wspace=1)
321
322 ax = plt.subplot(gs1[0, 0])
323 ax.plot(x, Exact_h[:, 50], 'b-', linewidth=2, label='Exact')
324 ax.plot(x, H_pred[50, :], 'r--', linewidth=2, label='Prediction')
325 ax.set_xlabel('$x$')
326 ax.set_ylabel('$|h(t,x)|$')
327 ax.set_title('$t = %.2f$' % (t[50, 0]), fontsize=10)
328 ax.axis('square')
329 ax.set_xlim([-5.1, 5.1])
330 ax.set_ylim([-0.1, 3.1])
331
332 ax = plt.subplot(gs1[0, 1])
333 ax.plot(x, Exact_h[:, 150], 'b-', linewidth=2, label='Exact')
334 ax.plot(x, H_pred[150, :], 'r--', linewidth=2, label='Prediction')
335 ax.set_xlabel('$x$')
336 ax.set_ylabel('$|h(t,x)|$')
337 ax.set_title('$t = %.2f$' % (t[150, 0]), fontsize=10)
338 ax.axis('square')
339 ax.set_xlim([-5.1, 5.1])
340 ax.set_ylim([-0.1, 3.1])
341
342 ax = plt.subplot(gs1[0, 2])
343 ax.plot(x, Exact_h[:, 250], 'b-', linewidth=2, label='Exact')

```

```

343 ax.plot(x, H_pred[250, :], 'r--', linewidth=2, label='Prediction')
344 ax.set_xlabel('$x$')
345 ax.set_ylabel('$|h(t,x)|$')
346 ax.axis('square')
347 ax.set_xlim([-5.1, 5.1])
348 ax.set_ylim([-0.1, 3.1])
349 ax.set_title('$t = \%.2f$' \% (t[250, 0]), fontsize=10)
350
351
352 gs2 = gridspec.GridSpec(2, 2)
353 gs2.update(top=.4, bottom=0, left=0.1, right=0.9, wspace=1)
354
355 ax = plt.subplot(gs2[1, 0])
356 ax.plot(x, Exact_h[:, 350], 'b-', linewidth=2, label='Exact')
357 ax.plot(x, H_pred[350, :], 'r--', linewidth=2, label='Prediction')
358 ax.set_xlabel('$x$')
359 ax.set_ylabel('$|h(t,x)|$')
360 ax.set_title('$t = \%.2f$' \% (t[350, 0]), fontsize=10)
361 ax.axis('square')
362 ax.set_xlim([-5.1, 5.1])
363 ax.set_ylim([-0.1, 3.1])
364
365 ax = plt.subplot(gs2[1, 1])
366 ax.plot(x, Exact_h[:, 450], 'b-', linewidth=2, label='Exact')
367 ax.plot(x, H_pred[450, :], 'r--', linewidth=2, label='Prediction')
368 ax.set_xlabel('$x$')
369 ax.set_ylabel('$|h(t,x)|$')
370 ax.axis('square')
371
372 ax.set_xlim([-5.1, 5.1])
373 ax.set_ylim([-0.1, 3.1])
374 ax.set_title('$t = \%.2f$' \% (t[450, 0]), fontsize=10)
375 ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.8), ncol=5,
376         frameon=False)
377 plt.savefig(f'grafica red neuronal.pdf', dpi=800, bbox_inches="tight")
378
379 # Plotting the losses for Adam and BFGS
380 plt.figure(figsize=(10, 5))
381 plt.subplot(2, 1, 1)
382 plt.plot(model.losses_adam, label='Adam Loss', color='b')
383 plt.xlabel('Iteration')
384 plt.ylabel('Loss')
385 plt.title('Losses of Adam optimizer')
386 plt.ylim(0, 1)
387 plt.legend()
388 plt.grid(True)
389 plt.tight_layout()
390
391 plt.subplot(2, 1, 2)
392 plt.plot(model.losses_bfgs, label='BFGS Loss', color='g')
393 plt.xlabel('Iteration')
394 plt.ylabel('Loss')
395 plt.title('Losses of BFGS optimizer')
396 plt.ylim(0, 1*10**(-3))
397 plt.legend()
398 plt.grid(True)
399 plt.tight_layout()

```

```

399     plt.savefig('losses_adam_bfgs.pdf', dpi=800)
400
401 plt.figure(figsize=(18, 12))
402
403 # Plotting exact u(t, x)
404 plt.subplot(3, 3, 1)
405 plt.pcolor(T, X, U_star, cmap='jet')
406 plt.colorbar()
407 plt.xlabel('$t$')
408 plt.ylabel('$x$')
409 plt.title('Exact u(t, x)')
410 plt.tight_layout()
411
412 # Plotting exact v(t, x)
413 plt.subplot(3, 3, 2)
414 plt.pcolor(T, X, V_star, cmap='jet')
415 plt.colorbar()
416 plt.xlabel('$t$')
417 plt.ylabel('$x$')
418 plt.title('Exact v(t, x)')
419 plt.tight_layout()
420
421 # Plotting exact h(t, x)
422 plt.subplot(3, 3, 3)
423 plt.pcolor(T, X, H_star, cmap='jet')
424 plt.colorbar()
425 plt.xlabel('$t$')
426 plt.ylabel('$x$')
427 plt.title('Exact h(t, x)')
428 plt.tight_layout()
429
430 # Plotting predicted u(t, x)
431 plt.subplot(3, 3, 4)
432 plt.pcolor(T, X, U_pred, cmap='jet')
433 plt.colorbar()
434 plt.xlabel('$t$')
435 plt.ylabel('$x$')
436 plt.title('Predicted u(t, x)')
437 plt.tight_layout()
438
439 # Plotting predicted v(t, x)
440 plt.subplot(3, 3, 5)
441 plt.pcolor(T, X, V_pred, cmap='jet')
442 plt.colorbar()
443 plt.xlabel('$t$')
444 plt.ylabel('$x$')
445 plt.title('Predicted v(t, x)')
446 plt.tight_layout()
447
448 # Plotting predicted h(t, x)
449 plt.subplot(3, 3, 6)
450 plt.pcolor(T, X, H_pred, cmap='jet')
451 plt.colorbar()
452 plt.xlabel('$t$')
453 plt.ylabel('$x$')
454 plt.title('Predicted h(t, x)')
455 plt.tight_layout()

```

```

456
457 # Plotting absolute error for u(t, x)
458 plt.subplot(3, 3, 7)
459 plt.pcolor(T, X, np.abs(U_star - U_pred), cmap='jet')
460 plt.colorbar()
461 plt.xlabel('$t$')
462 plt.ylabel('$x$')
463 plt.title('Absolute error for u(t, x)')
464 plt.tight_layout()
465
466 # Plotting absolute error for v(t, x)
467 plt.subplot(3, 3, 8)
468 plt.pcolor(T, X, np.abs(V_star - V_pred), cmap='jet')
469 plt.colorbar()
470 plt.xlabel('$t$')
471 plt.ylabel('$x$')
472 plt.title('Absolute error for v(t, x)')
473 plt.tight_layout()
474
475 # Plotting absolute error for h(t, x)
476 plt.subplot(3, 3, 9)
477 plt.pcolor(T, X, np.abs(H_star - H_pred), cmap='jet')
478 plt.colorbar()
479 plt.xlabel('$t$')
480 plt.ylabel('$x$')
481 plt.title('Absolute error for h(t, x)')
482 plt.tight_layout()
483 plt.savefig('comparacion.pdf')

```

Bibliografia

- [1] Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer 2006.
- [2] Physics Informed Deep Learning (Part I): Data-driven Solutions of Non-linear Partial Differential Equations Maziar Raissi, Paris Perdikaris , and George Em Karniadakis, arXiv:1711.10561v1 [cs.AI] 28 Nov 2017
- [3] Michael Nielsen, Neural Networks and Deep Learning, 2019
- [4] Mark J. Ablowitz, David J. Kaup, Alan C. Newell and Harvey Segur, Nonlinear-Evolution Equations of Physical Significance, Phys. Rev. Lett. 31, 125 (1973).
- [5] Hsing-Hen Chen, General Derivation of Bäcklund Transformations form Inverse Scatterin Problems, Phys. Rev. Lett. 33, 925 (1974).
- [6] Abbas A. Rangwala and Jyoti A. Rao, Complete soliton Solutions of the ZS / AKNS Equations of the Inverse Scatterin Method, Phys. Rev. Lett. 112A, 188 ,(1985).
- [7] V.N. Serkin, Akira Hasegawa and T.L. Belyaeva, Solitary waves in nonautonomous nonlinear and dispersive systems: nonautonomous solitons, Journal of Modern Optics 57, 1456–1472, (2010).
- [8] T. L. Belyaeva and V. N. Serkin, Nonautonomous Solitons: Applications from Nonlinear Optics to BEC and Hydrodynamics, Hydrodynamics – Advanced Topics, (2011).
- [9] STEFANO MARKIDIS, The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers?, arXiv:2103.09655v2 [math.NA] 5 Jul 2021
- [10] Zijian Zhou and Zhenya Yan, Deep learning neural networks for the third-order nonlinear Schrodinger equation: Solitons, breathers, and rogue waves, arXiv:2104.14809v1 [nlin.PS] 30 Apr 2021
- [11] Maziar Raissi and George Em Karniadakis, Hidden Physics Models: Machine Learning of Nonlinear Partial Differential Equations, arXiv:1708.00588v2 [cs.AI] 21 Aug 2017
- [12] Jun-Cai Pu, Jun Li and Yong Chen, Soliton, breather, and rogue wave solutions for solving the nonlinear Schrodinger equation using a deep learning method with physical constraints, Chin. Phys. B Vol. 30, No. 6 (2021) 060202.
- [13] P.H. Winston (1992) , Artificial Intelligence Addison-Wesley, Massachusetts, USA.

- [14] Artificial Intelligence. A Modern Approach (2010) Third Edition, S. Russell, P. Norvig, Editors, Pearson Education, New Jersey.
- [15] <https://www.oracle.com/mx/artificial-intelligence/what-is-ai/>
- [16] C. M. Bishop (2006), Pattern Recognition and Machine Learning, Springer.
- [17] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *Nature* 521, 436 (2015).
- [18] M. Raissi (2019), Hidden physics models: Machine learning of nonlinear partial differential equations.
- [19] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, *Commun. Numer. Methods Eng.* 10, 195-201 (1994).
- [20] I. E. Lagaris, A. Likas, and D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Trans. Neural Networks* 9, 987-1000 (1998).
- [21] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, *J. Comput. Phys.* 348, 683-693 (2017).
- [22] M. Raissi, G. E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, *J. Comput. Phys.* 357 (2018) 125-141.
- [23] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, *J. Comput. Phys.* 375, 1339-1364 (2018).
- [24] M. Raissi, P. Perdikaris, G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.* 378 (2019) 686-707.
- [25] M. Raissi, A. Yazdani, G. E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, *Science* 367 (2020) 1026-1030.
- [26] A. Jagtap and G. Karniadakis (2020), Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, *Communications in Computational Physics*, 28, 2002-2041.
- [27] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis (2021), Deep xde: A deep learning library for solving differential equations, *SIAM Review*, 63(1), 208228, doi: 10.1137/19m1274067.
- [28] Z. Zhou and Z. Yan, Solving forward and inverse problems of the logarithmic nonlinear Schrödinger equation with PT-symmetric harmonic po-

- tential via deep learning, *Phys. Lett. A* 387 (2021) 127010.
- [29] L. Wang and Z. Yan, Data-driven rogue waves and parameter discovery in the defocusing NLS equation with a potential using the PINN deep learning, *arXiv: 2012.09984* (2020).
- [30] I. Bilonis (2021), A hands-on introduction to physics-informed machine learning.
- [31] L. McClenny and U. Braga-Neto (2020), Self-adaptive physics-informed neural networks using a soft attention mechanism. *arXiv:2009.04544*
- [32] G. Marcucci, D. Pierangeli, C. Conti, Theory of neuromorphic computing by waves: machine learning by rogue waves, dispersive shocks, and solitons, *Phys. Rev. Lett.* 125, 093901 (2020).
- [33] Z. Zhou, Zh. Yan (2021) Deep learning neural networks for the third-order nonlinear Schrödinger equation: Solitons, breathers, and rogue waves, *Commun. Theor. Phys.* 73 105006.
- [34] D. PIERANGELI, G. MARCUCCI , C.CONTI, Photonic extreme learning machine by free-space optical propagation, *Photonics Research* 9 (2021) 1446
- [35] M. Nakajima, K. Tanaka, T. Hashimoto (2022) Neural Schrödinger Equation:Physical Law as Neural Network, *arXiv:2006.13541*
- [36] M. A. Ablowitz, P. A. Clarkson, Solitons, nonlinear evolution equations and inverse scattering, Cambridge, 1991.
- [37] A. Hasegawa, Optical Solitons in Fibers, Springer-Verlag, Berlin, 1989.
- [38] G. P. Agrawal, Nonlinear Fiber Optics, 3 rd edition, Academic Press, 1995.
- [39] T.L. Belyaeva, M. A. Agüero, Nonautonomous solitons of the novel nonlinear Schrödinger equation: Self-compression, amplification, and the bound state decay in external potentials, *Optik* 244 (2021) 167584.
- [40] Chen, H. H., & Liu, C. S. (1976). Solitons in nonuniform media. *Physical Review Letters*, 37, 693.